

Bachelor final project

Bachelor's degree in Industrial Technology Engineering

**Control design and implementation
of a twin robot**

Author: Mireia Perna Vila
Director: Arnau Dòria Cerezo
Víctor Repecho del Corral
Convocatòria: June 2018



Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona



SUMMARY

The purpose of this project is the creation of a controller for an autonomous robot. To achieve this, the accelerometer of the STM32F4-Discovery board will be used which will be coupled to the robot.

This project is an extension of previous projects referenced below which worked with the same robot implementing the following application:

- Design and implementation of a line tracking system for the robot.
- Design and implementation of a WiFi communication system for the robot.

The extension will be the design and implementation of a PD controller for the line tracking robot. The innovation will be the use of the acceleration data obtained directly from the accelerometer MEMS of the discovery board in order to design the derivative controller.

First step will be the theoretical design of the controller using the Routh-Hurwitz theorem and the following simulations using Matlab and Simulink, to prove the response of the robot that will be expected in the experimental tests.

Once the controller is designed, the discovery board will be programmed in order to implement the controller. To condition the data acquired for the accelerometer it is going to be designed a first order low-pass filter and an axis rotation system in order to use this data for the controller.

Finally, using a communication system and the line tracking system it is going to be proved the real response of the robot and the controller by experimental tests in a closed circuit.

CONTENTS

SUMMARY	3
CONTENTS	4
1. Introduction.....	6
1.1. Objectives and scope of the project.....	6
2. Control design.....	7
2.1. Model.....	7
2.2. Analysis and control tuning	8
2.2.1. Stability	10
2.2.2. Pole location	11
2.3. Simulations.....	12
2.3.1. Implication of the parameters in the response.....	13
2.3.2. Tuning of K_P and K_D according to the system requirements.....	16
3. Hardware and software description	19
3.1. STM32F4-Discovery Board	19
3.1.1. Accelerometer LIS3DSH.....	20
3.1.2. User and reset buttons.....	21
3.2. WiFi access point and network router	21
3.3. Software	22
3.3.1. Eclipse and System Workbench (for STM32).....	22
3.3.2. Viewers	22
3.3.2.1. STMStudio	22
3.3.2.2. Python application - Python (x,y).....	23
4. Signal conditioning.....	24
4.1. Low-pass filter design	24
4.2. Axes rotation system	28
5. Experimental tests	31
CONCLUSIONS.....	33
BUDGET	34

ACKNOWLEDGMENTS.....	35
BIBLIOGRAPHY	36
ANNEXE: Software Code	37

1. Introduction

1.1. Objectives and scope of the project

The main objective of the project is to design and implement a PD controller in an autonomous vehicle using the development board STM32FA-Discovery. The idea is to use the acceleration data from the accelerometer MEMS as the input signal of the derivative controller.

In order to implement the controller in the board, because of the predetermined conditions of the accelerometer, values obtained directly are not correct since is affected by disturbances of the environment (noise), the offset on each axis and the orientation of the board respect the vehicle, as it could not be completely parallel to this giving incorrect values. Therefore, it is necessary to implement a low pass filter to remove noise signal and a rotation system for the axis so the z component is always aligned with gravity.

Once the controller is designed and the simulations has been done, the objective of the project is to compare the response of the system obtained in the simulations with the real response of the robot obtained in experimental tests.



Figure 1 – STM32-Discovery Board

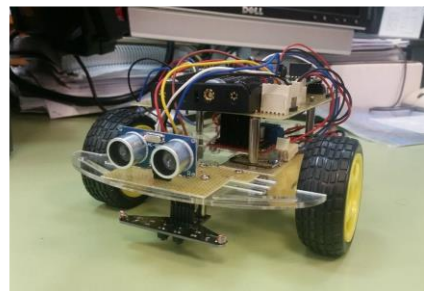


Figure 2 – Vehicle Robot

In order to achieve it previous projects have been used from where it has been used the kinematic and dynamic studies of the system and the line tracking system from [1] in order to design the new controller and the experimental test, and from [2] the communication system in order to prove the response of the robot.

The final point of the work is the comparison of the response that is to be obtained according to the calculated parameters and the actual response, since the model of system used is a very simplified model that does not take into account frictions, parameters of the motor and others that alter the behaviour of the vehicle.

2. Control design

2.1. Model

As mentioned above, the model proposed is according to the line tracking study from [1] where a PI controller is used to perform this control. The kinematic model used is the following

$$u_1 = \frac{r}{2}(\omega_R + \omega_L) \quad (1)$$

$$u_2 = \frac{r}{2 \cdot R}(\omega_R - \omega_L) \quad (2)$$

Where u_1 is the linear speed, u_2 is the angular speed of the vehicle and ω_R and ω_L are both the angular speed of the right (R) and left (L) wheels. The parameters r and R are from the vehicle design and presented in the following table

PARAMETER	MEANING	MEASUREMENT [SI]
R	Distance between the wheels	0.17
r	Radius of the driving wheels	0.035
l	Distance from the axle to the tracking sensor	0.068
l_a	Distance from the axle to the accelerometer	0.09

TABLE 1 – VALUES OF VEHICLE PARAMETERS

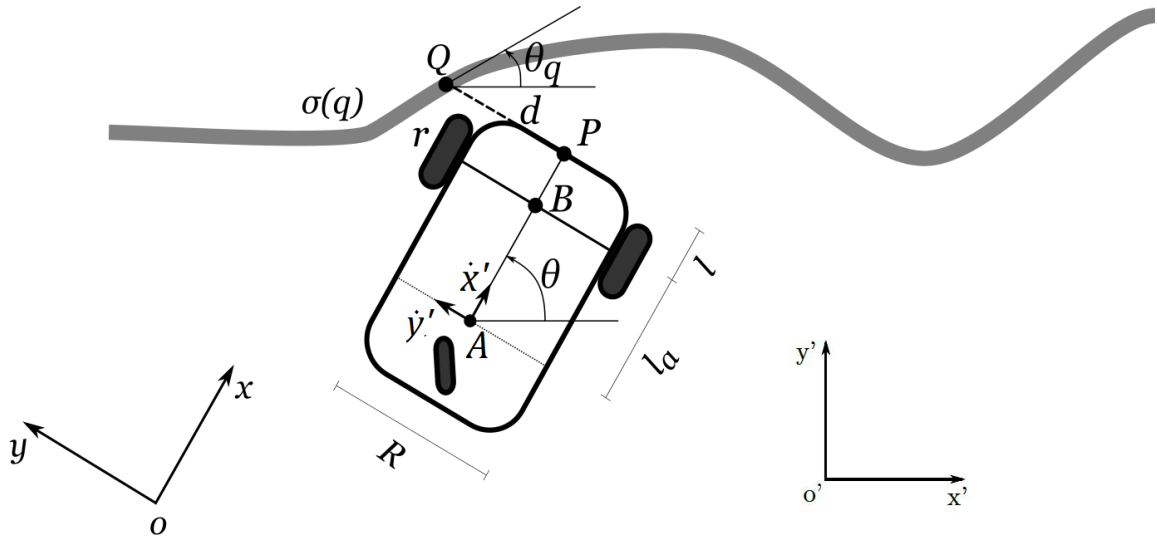


Figure 3 – Scheme of the vehicle and the line that follows

Other variable parameters that must be taken into account are d which is the distance of the sensor to the line, θ which is the angle formed by the axes of the vehicle with the real axes $o'x'y'$, $\sigma(q)$ which is the curvature of the line in each point and θ_q which is the angle formed by the direction of the line and the real axes.

On the other hand, the points P , B , A and Q correspond respectively to the line tracking sensor, the middle point of the axis of the wheels, the accelerometer and the point of the line to which it is directed.

It should be taken into account that throughout the project it has worked with a linear speed u_1 of $0.18m/s$.

As a two-wheeled vehicle is used, the rotation centre it is located in the middle of the wheel axel so the kinematic model of it with respect to this point B is given by the following system

$$\begin{cases} \dot{x}' = \cos(\theta) u_1 \\ \dot{y}' = \sin(\theta) u_1 \\ \dot{\theta} = -u_2 \end{cases} \quad (3)$$

After determining the desired trajectory followed by the vehicle and linearizing the system, calculated in [1] the following transfer function of the plant of the system is obtained

$$G(s) = \frac{D(s)}{U_2(s)} = \frac{ls + v\alpha}{s^2 + v^2c^2} \quad (4)$$

Where $\alpha = \sqrt{1 - l^2c^2}$, the input signal is the angular speed u_2 and the controlled parameter is the distance to the line d . The parameter l is defined in the table above, v is the linear speed u_1 and c is the curvature defined as $1/R_{curv}$, where R_{curv} is the radius of the curve of the path followed.

2.2. Analysis and control tuning

Parting from the system model from [1] showed as follow

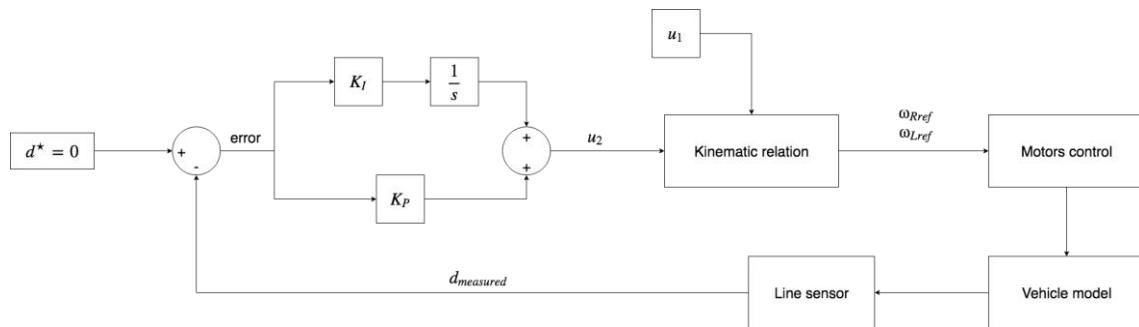


Figure 4 – Scheme of control of the trajectory taken from [1]

The line tracking control is a PI controller that takes as the control parameter the distance between the vehicle sensor to the line of the circuit, so as it is considered the reference distance d_{ref} equal to zero, the error signal e_1 corresponds to

$$e_1 = d_{ref} - d_{measured} = -d_{measured} \quad (5)$$

As it is known that the derivative action gives a better response it is been decided to implement a PD controller, where the derivative signal will be the data take it directly from the accelerometer so a causality problem it is avoid. The proportional action will take as input signal the same as before.

It must be taken into account that we need speed data in order to implement the derivative part so as the accelerometer gives acceleration data, it is necessary to incorporate an integrator element.

The idea is that the speed in y axis v_y (angular speed), given by the accelerometer data is transferable to the speed of the tracking sensor since the vehicle is working as a rigid as follows

$$\dot{d} = -\frac{l}{l_a} v_y \quad (6)$$

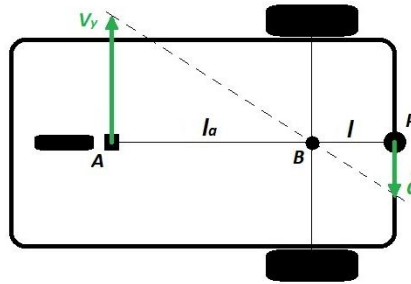


Figure 5 – Scheme of the kinematic relation (6)

Where l_a the distance between the accelerometer and the rotation is B , and l is the distance between the rotation centre and the sensor of the line tacking. These values are determined in Table 1.

So as it is considered the reference value of it equal to zero $\dot{d}_{ref} = 0$, the error signal e_2 corresponds to

$$e_2 = \dot{d}_{ref} - \dot{d} = -\dot{d} \quad (7)$$

So u_2 will be determined by the following signal

$$u_2 = K_P \cdot e_1 + K_D \cdot \frac{de_2}{dt} = K_P \cdot (-d_{measured}) + K_D \cdot \frac{d(-\dot{d})}{dt} \quad (8)$$

Where K_P is the proportional gain and K_D the derivative gain.

Finally, the transfer function of the controller and the final model is the following

$$C(s) = K_P + K_D s \quad (9)$$

$$W(s) = \frac{C(s) \cdot G(s)}{1 + C(s) \cdot G(s)} = \frac{(K_P + K_D s)(ls + v\alpha)}{(s^2 + v^2 c^2) + (K_P + K_D s)(ls + v\alpha)} \quad (10)$$

And the bloc diagram of the system

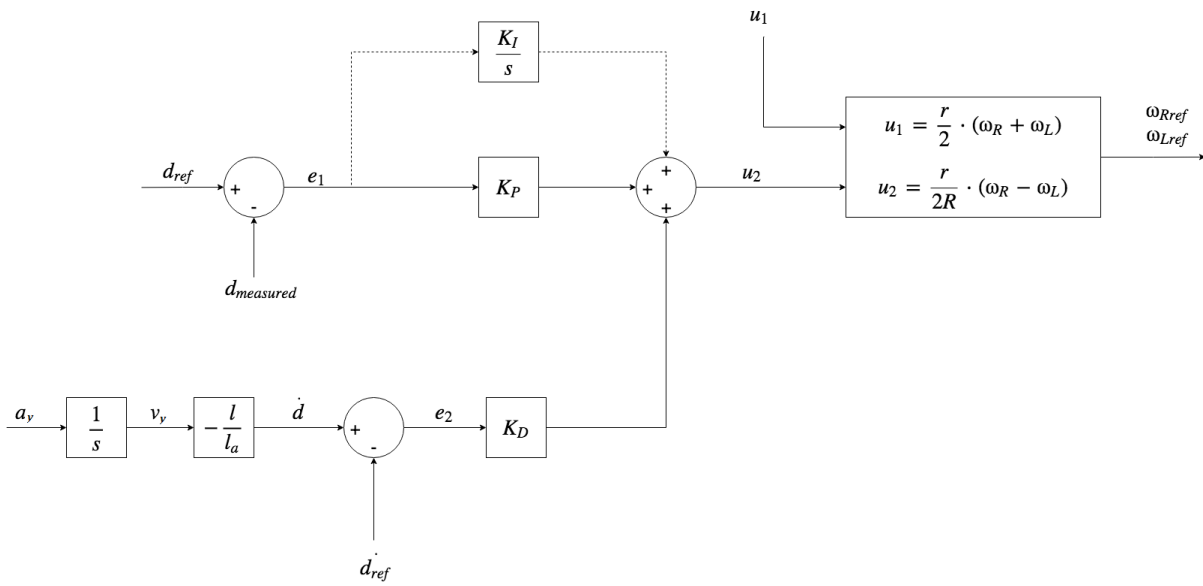


Figure 6 – Bloc diagram of the controller

2.2.1. Stability

Next step is to determinate the values of the gains K_P and K_D in order to impose a stable system. For this it is used the Routh-Hurwitz theorem to find the ranges of the gains that meet this need.

The characteristic polynomial is given by

$$\begin{aligned} Q(s) &= 1 + C(s) \cdot G(s) = (1 + K_D l)s^2 + (K_P l + K_D v\alpha)s + (v^2 c^2 + K_P v\alpha) \\ &= s^2 + \frac{(K_P l + K_D v\alpha)}{(1 + K_D l)}s + \frac{(v^2 c^2 + K_P v\alpha)}{(1 + K_D l)} \end{aligned} \quad (11)$$

The parameters that accompany each grade are the ones needed to implement this method which ordered from higher to lower grade are $a_2 = 1$, $a_1 = \frac{(K_P l + K_D v\alpha)}{(1 + K_D l)}$ and $a_0 = \frac{(l^2 c^2 + K_P v\alpha)}{(1 + K_D l)}$.

The theorem says that the number of sign changes between the coefficients of the first column corresponds to the number of positive roots of the system, so there must be no sign changes since for a system to be stable all roots must have a negative real part

$$\begin{array}{c|cc} s^2 & a_2 & a_0 \\ s^1 & a_1 & 0 \\ s^0 & b_1 & b_2 \end{array} \quad (12)$$

Where $b_1 = -\frac{1}{a_1} \begin{vmatrix} a_2 & a_0 \\ a_1 & 0 \end{vmatrix} = a_0$ and $b_2 = -\frac{1}{a_1} \begin{vmatrix} a_2 & 0 \\ a_1 & 0 \end{vmatrix} = 0$

Imposing that parameters from the first column must be positive, the following relation is found

$$K_P l + K_D v \alpha > 0 \quad (13)$$

$$l^2 c^2 + K_P v \alpha > 0 \quad (14)$$

As it is known that the rest of parameters l, α, v, c defined before are always positive, the following restrictions are the ones to determine the stability of the system

$$K_P > \frac{-vc^2}{\alpha} \quad (15)$$

$$K_D > \frac{lc^2}{\alpha^2} \quad (16)$$

In case of following a straight line defined by $c = 0$, the restriction is $K_P > 0$ and $K_D > 0$.

2.2.2. Pole location

To obtain a stable system the poles of it must have negative real part. Since the characteristic equation is of second order, there are two imaginary poles conjugated according to

$$\begin{cases} p_1 = -\sigma + \omega j \\ p_2 = -\sigma - \omega j \end{cases} \quad (17)$$

Where $\sigma = \frac{4}{t_s}$ and $\omega = \frac{-\pi\sigma}{\ln(M_p)}$

The parameter t_s is the settling time defined as the time where the response reaches the 98% of its value and M_p is the maximum overshoot defined as the maximum peak value of the response measured from the desired response of the system. These parameters are imposed depending on the desired response of the system.

In order to determine the values of K_P and K_D , as it is known that the condition of stability is determined by its poles, it is imposed a desired characteristic polynomial $Q'(s)$ as follows

$$Q'(s) = (s - p_1)(s - p_2) = s^2 + (-p_1 - p_2)s + (p_1 p_2) \quad (18)$$

Imposing $Q'(s) = Q(s)$ and equalizing the parameters that accompany each grade as follows

$$a_1 = -p_1 - p_2 = 2\sigma \quad (19)$$

$$a_0 = p_1 p_2 = \sigma^2 + \omega^2 \quad (20)$$

It is obtained a relation between the value of the gains and the parameters of the poles determined by

$$K_D = \frac{2\sigma - K_P l}{v\alpha - 2\sigma l} \quad (21)$$

$$K_P = \frac{2\sigma l \gamma + \beta(1 - v^2 c^2)}{\gamma l^2 + \beta v \alpha} \quad (22)$$

Where $\gamma = \sigma^2 + \omega^2$ and $\beta = v\alpha - 2\sigma l$

With this relation it is possible to determine the optimum value of the gains while the stability condition is met.

It must be taken into account that these values of K_P and K_D are considering that there are no zeros (roots in the numerator) which is not the case. When doing the simulations with these values, a different response to the one imposed with t_s and M_p will be observed due to the influence of the zeros, so the gains will have to be adjusted in order to meet the desired requirements.

2.3. Simulations

In the following section different simulations are done in order to adjust the value of the gains of the controller, using Matlab and Simulink.

To model the plant of the system in Matlab it has been used the linearized form from [1], to make it easier to work with. This model has the following structure

$$G(s) = C(sI - A)^{-1}B \quad (23)$$

Where $A = \frac{v}{\alpha} \begin{pmatrix} -c^2 l & -1 \\ c^2 l & -1 \end{pmatrix}$, $B = \begin{pmatrix} l \\ -1 \end{pmatrix}$ and $C = (1 \ 0)$

Once the plant and all the parameters are defined in Matlab with K_P and K_D as in (21) and (22), the system is modelled in Simulink adding visualizers in time, u_2 and d response it is possible to graph these signals in order to study them.

The following is the Simulink model

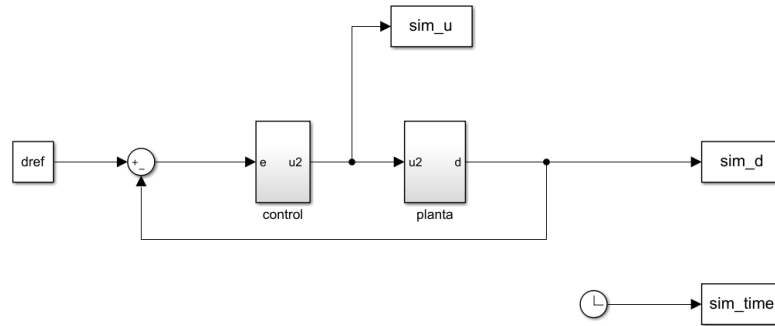


Figure 7 – Simulink model of the system

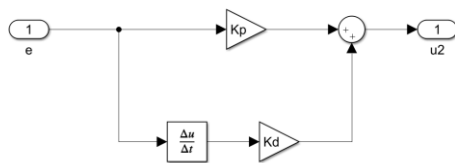


Figure 8 – Simulink model of the controller

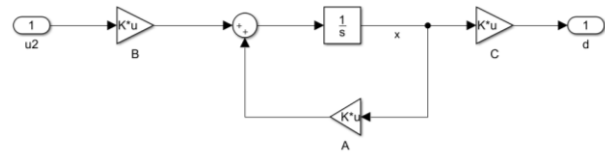


Figure 9 – Simulink model of the plant

2.3.1. Implication of the parameters in the response

In the first place, a study has been carried out to see how the variation of the variable parameters of the system affects the response obtained.

These are the curvature of the line followed c , the settling time t_s and the overshoot M_p of the system, but the settling time and the overshoot are parameters imposed at the beginning according to the conditions of the system and the proposed requirements, however the curvature can vary throughout the study depending on the path followed. Therefore to carry out the simulations different values of t_s and M_p have been imposed and from there it has been observed how the curvature affects the system.

The maximum value of the curvature that Matlab accept is 14.7, so the simulations performed took five different values of c between 0 (straight line) and 14.

- $t_s = 0.5s$ and $M_p = 5\%$

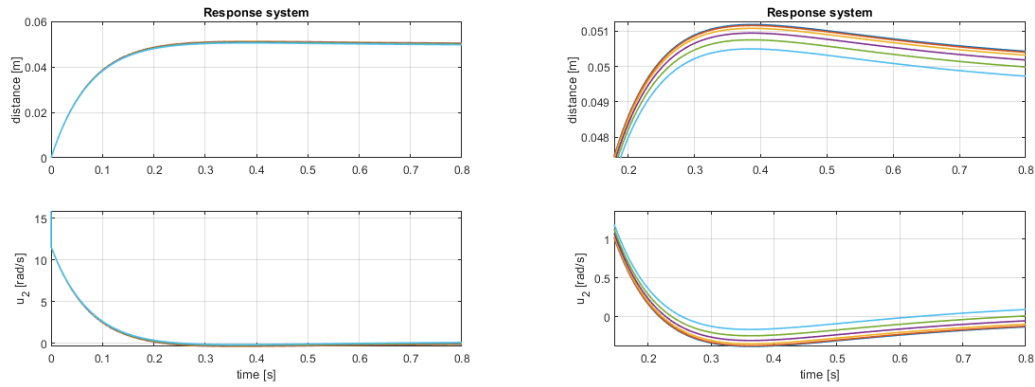


Figure 10 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 0.5s$ and $M_p = 80\%$

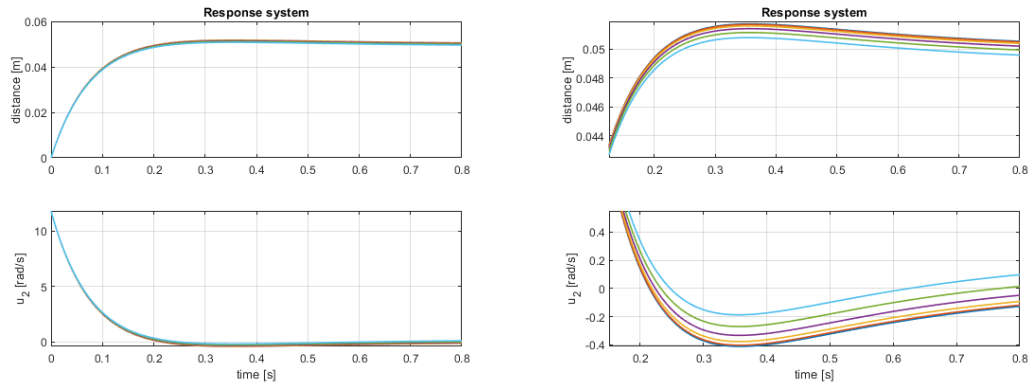


Figure 11 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 2s$ and $M_p = 5\%$

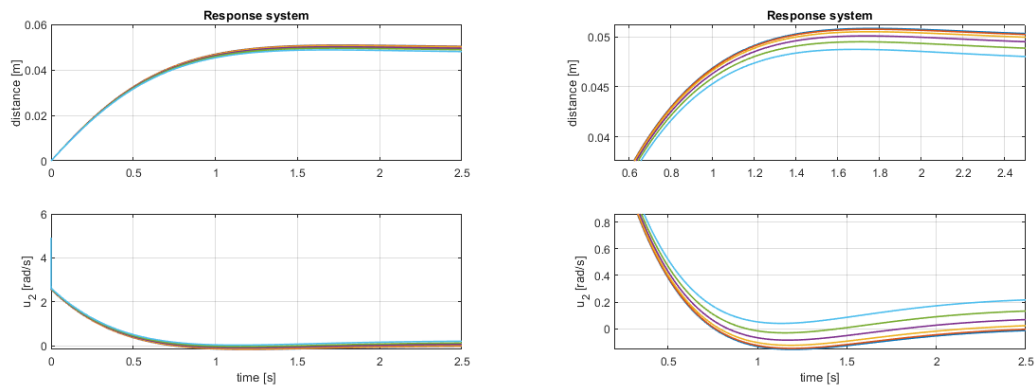


Figure 12 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 2s$ and $M_p = 80\%$

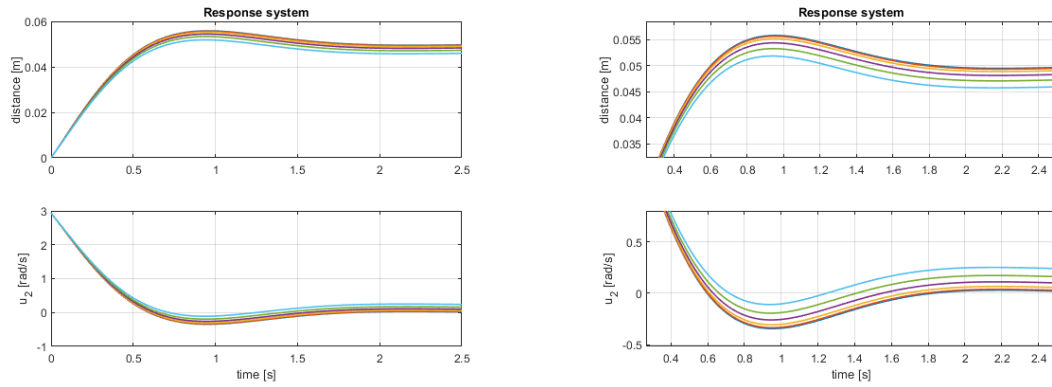


Figure 13 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 5s$ and $M_p = 5\%$

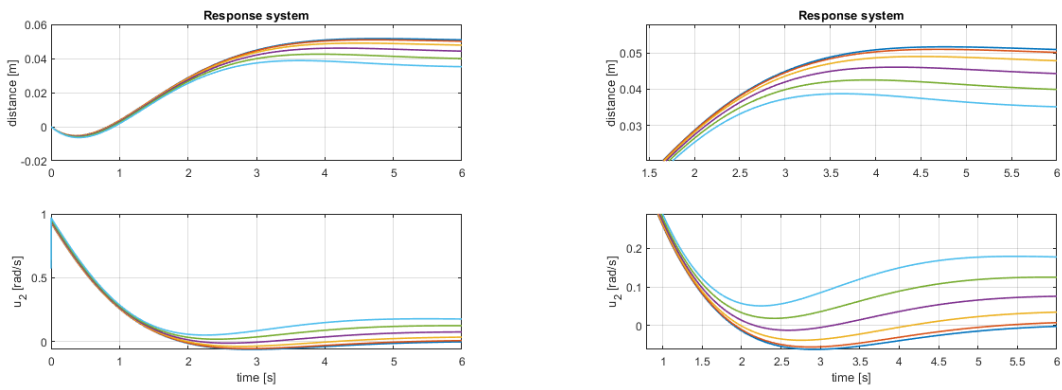


Figure 14 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 8s$ and $M_p = 5\%$

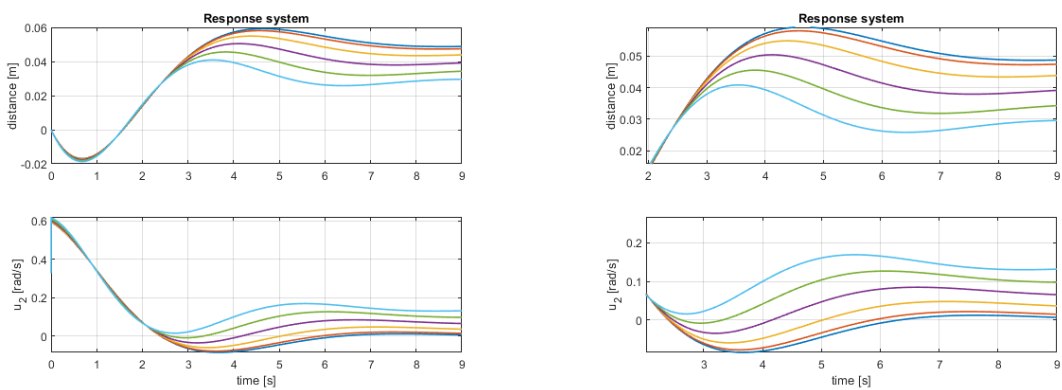


Figure 15 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 8s$ and $M_p = 80\%$

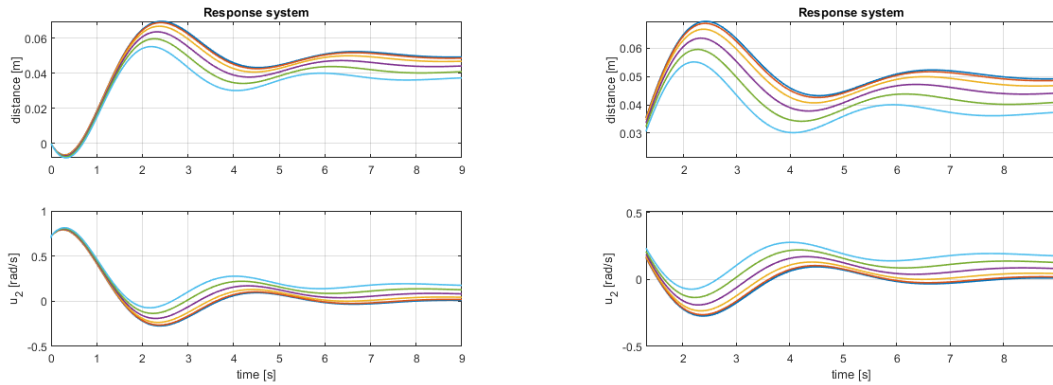


Figure 16 – Simulation (left) and zoom of the stationary state (right)

These simulations show that the effect of the curvature of the path followed it is more significant as the settling time and the overshoot increase, but as this parameter is an external factor it is not possible to control, so adjusting the values of t_s and M_p according to the requirements of the system, the design of the controller will be done.

2.3.2. Tuning of K_P and K_D according to the system requirements

To meet the requirements of the system it has been determined the maximum speed that can reach the wheels of the vehicle, since it is equivalent to the maximum power of the engine.

It has been measured with the charged batteries (5V) and when the vehicle is both in contact and not in contact with the ground.

In the following graphs it can be observed that there is not an important difference between the vehicle being in contact or in non-contact, but since the studies are carried out with the vehicle in contact with the ground, the maximum speed is sought in this state.

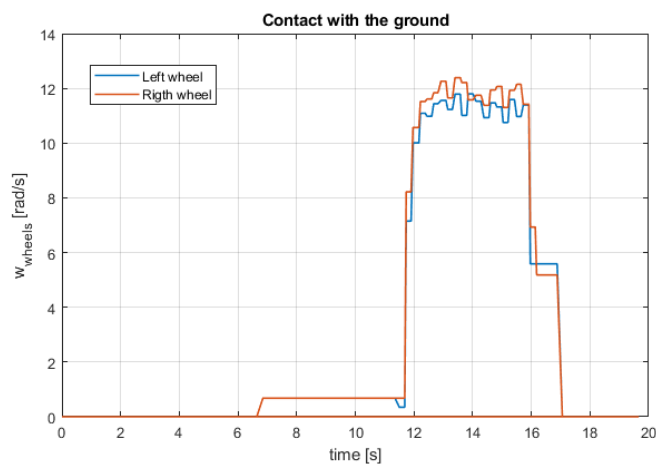


Figure 17 – Wheels speed in contact with the ground

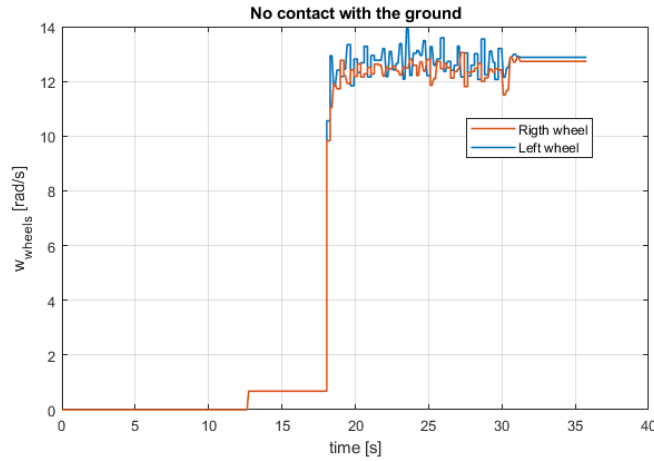


Figure 18 – Wheels speed in no contact with the ground

It must be taken into account that when making the tests in contact with the ground, the data collection is limited to the path that the vehicle can follow. As the tests were done in the laboratory, the duration of the test is shorter because when encountering obstacles, the robot has a device that makes it stop before crashing, so at the end of each test there is a peak fall in speed.

The maximum values obtained in the contact test are 11.8 rad/s for left wheel and 12.39 rad/s for the right wheel.

Once the maximum speed that each wheel can reach has been found, it can be limited the range of angular speed u_2 in which the vehicle can work according to the kinematic relation of the system (1) and (2) as follows

$$\begin{aligned} u_2 &< \frac{1}{R} (r\omega_{Rmax} - u_1) \\ u_2 &> \frac{1}{R} (u_1 - r\omega_{Lmax}) \end{aligned} \quad (24)$$

The maximum speed values of each wheel are different but in motion both wheels must have the same speed to follow a straight line, therefore the angular speed range u_2 of the vehicle is determined taking the minimum between the maximum speeds measured as the maximum that can be achieved at the same time. So $\omega_{Rmax} = \omega_{Lmax} = \min(11.8 ; 12.39) = 11.8 \text{ rad/s}$

Knowing these restrictions, the appropriate value of the gains K_P and K_D can be determined by checking the response through simulations. The main point of these simulations is to verify that the maximum value of the response, which is the angular speed, is in the rank determined by

$$-1.3706 \text{ rad/s} < u_2 < 1.3706 \text{ rad/s}$$

With the previous simulations it has been possible to determine that the angular speed u_2 decreases as the overshoot decreases and increases the settling time.

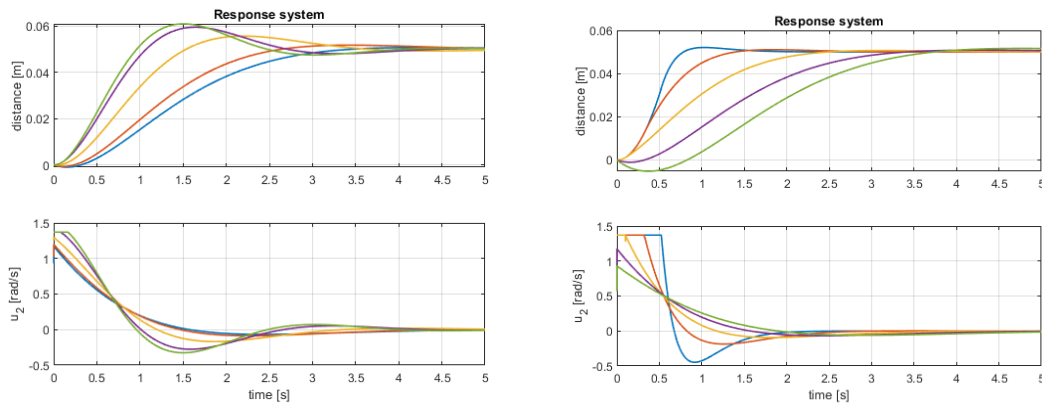


Figure 19 – Response when M_p varies (left), response when t_s varies (right)

In the following simulations it is imposed $c = 0$ and the maximum angular speed $u_2 = 1.3706 \text{ rad/s}$, so it is easy to see the behaviour of the response when varying the overshoot (between 0 and 90%) and the settling time (between 0 and 5s) by separate.

Testing different values, doing the relevant simulations and taking into account that the overshoot is usually between 5 and 10% and that it is not desired to have a high settling time, it has been proposed the values $t_s = 3.6\text{s}$ and $M_p = 5\%$.

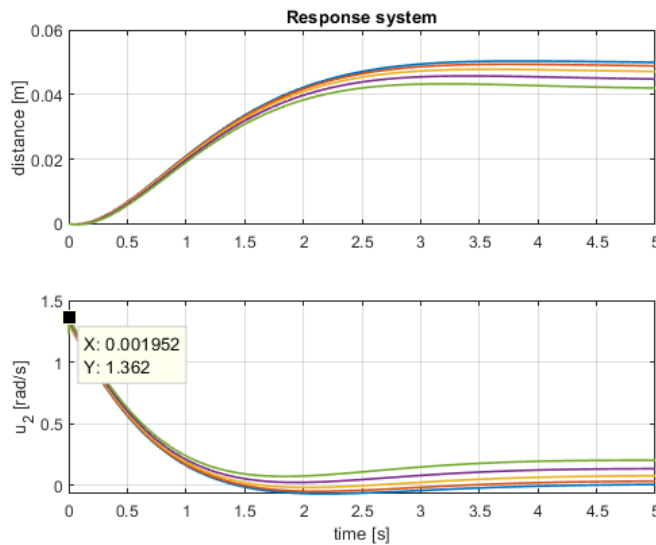


Figure 20 – Response for maximum u_2 desired

With these parameters the maximum speed required by the wheels is within the determined range. Varying the parameters minimally it is possible to get a speed closer to the maximum imposed but considering that these changes are hundreds of seconds it has been worked with these values, obtaining the following values for the gains $K_p = 24.4731$ and $K_D = 19.3172$.

3. Hardware and software description

3.1. STM32F4-Discovery Board

It is a development board aimed at both beginner and experienced users that allow creating and developing applications through a high performance microcontroller.

It offers the following features:

- STM32F407VGT6 microcontroller featuring:
 - 32-bit ARM Cortex-M4 with FPU core
 - 1 Mb Flash memory
 - 192 Kb RAM in a LQFP100 package
 - 168 MHz clock rate
- ST-LINK embedded tool for programing and debugging
- USB ST-LINK with re-enumeration capability and three different interfaces: Virtual COM port, mass storage and debug port.
- Board power supply through USB cable or an external 3V-5V power supply.
- LIS3DSH ST MEMS 3-axis accelerometer.
- MP45DT02 ST MEMS audio sensor omni-directional digital microphone.
- CS43L22 audio DAC with integrated class D speaker driver.
- Eight LEDs:
 - LD1 (red/green) for USB communication.
 - LD2 (red) for 3.3 V power on.
 - Four user LEDs: LD3 (orange), LD4 (green), LD5 (red), LD6 (blue).
 - 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current.
- Two push buttons: user and reset.
- USB OTG FS with micro-AB connector.
- Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing.
- Comprehensive free software including a variety of examples, part of the STM32CubeF4 package or STSW-STM32068 for legacy standard library usage.

In this project the main peripheral used is the accelerometer, still others are used such as the push buttons and the LEDs to control the operation of the device and the turning on and off of the system, and others used in [1] and [2].

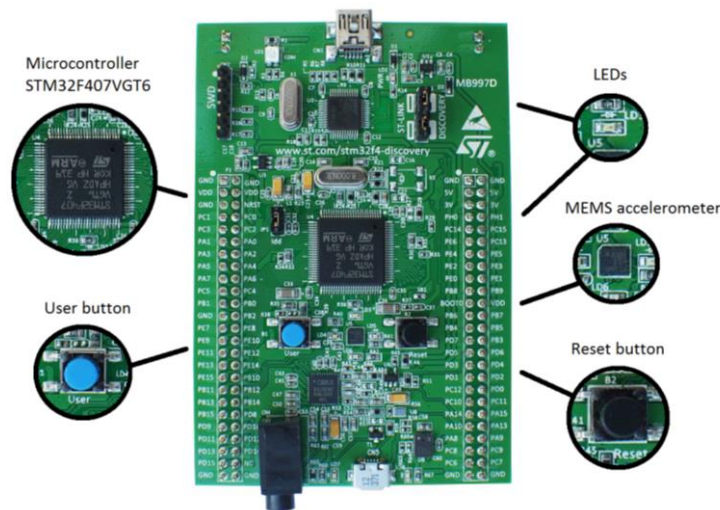


Figure 21 – STM32F4-Discovery Board

3.1.1. Accelerometer LIS3DSH

The accelerometer is the component LIS3DSH an ultra-low-power high-performance three-axis lineal accelerometer, which is able to pick up the acceleration in the three prefixed axis according to the *Figure 22*, and can be programmed to implement autonomous applications as motion-controlled user interface, gaming and virtual reality, pedometers, intelligent power saving for handheld devices, display orientation, click/double-click recognition, impact recognition and logging, vibration monitoring and compensation.

The features of the device are:

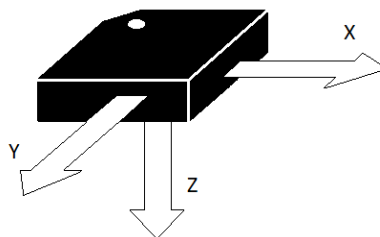


Figure 22 – Layout of the axes according to the accelerometer

- Wide supply voltage 1.71V-3.6V
- Independent IOs supply (1.8V) and supply voltage compatible
- Ultra-low power consumption
- $\pm 2g/\pm 4g/\pm 6g/\pm 8g/\pm 16g$ dynamically selectable full scale
- I²C/SPI digital output interface
- 16-bit data output

- Programmable embedded state machines
- Embedded temperature sensor
- Embedded self-test
- Embedded FIFO buffer
- 10000g high shock survivability
- ECOPACK, RoHS and 'Green' compliant

The units of measurement used are mg, where gravity is 1000mg. For this application the dynamically rank it is set to $\pm 2g$ in the three axes, as it is the lowest rank possible and the vehicle is not able to reach these values.

It must be taken into account that during the project all the kinematic relations have been made defining that the linear speed u_1 is on the x axis, so at the moment of coupling the plate to the vehicle, this has to be oriented so that the x axis of the accelerometer is in the direction of rectilinear movement of the vehicle.

3.1.2. User and reset buttons

User button is connected to I/O PA0 pin of the microcontroller and reset button to NRST which function is to reset the microcontroller.

The user button it can be given different uses according with the purpose. In this project it controls the start-up of the vehicle and the axes calibration system of the accelerometer which is explained later.

3.2. WiFi access point and network router

The WiFi control system comes from [2] which consisted of implementing a communication via WiFi between the vehicle and the computer. This communication allows carrying out tests with the vehicle without having to be connected by cable to the computer. The communication allows you to send orders from the computer and at the same time receive the desired information acquired by the board.

The WiFi module ESP8266 is used, which is directly coupled to the vehicle and allows WiFi connection. It is a low cost device that uses a serial port IP for sending data, and therefore can be implemented in many more devices. An important point is that the work of the device does not interfere with that of the microcontroller.

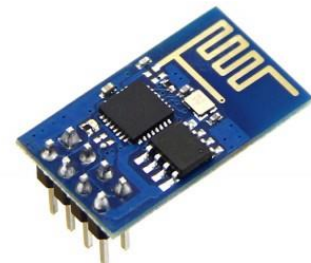


Figure 23 – WiFi module ESP8266

It is also necessary to use a WiFi access point to create the network to which the vehicles and the control computer will connect.

All this part is implemented and explained in [2], where the structure of the system is the following

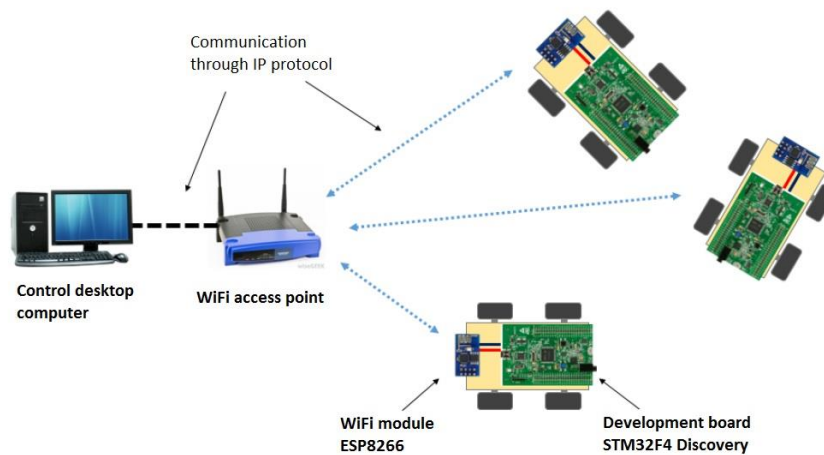


Figure 24 – WiFi connections from [2]

3.3. Software

3.3.1. Eclipse and System Workbench (for STM32)

Eclipse is an integrated development environment of code that allows developing projects in different languages as C, C++, Python and others as long as the connectors needed for each language are installed.

System Workbench is one of the Eclipse environments that provide a software development, compile and debug tools for STM32 boards and allow creating an integrated project for the development board used.

About this software it has been able to implement the project using code written in C language the previous projects [1] and [2].

In the annexes is the entire code.

3.3.2. Viewers

3.3.2.1. STMStudio

This program allows viewing and recording the behaviour of the parameters desired on the screen in real time but with the disadvantage that it can only work with the vehicle connected by cable to the computer.

It is a useful tool when determining parameters that do not require the continuous movement of the vehicle as the noise detected by the accelerometer and the deviation of the axes thereof.

It is an easy software to use and understand. It consists of a manual [6] which explains step by step the installation and configuration of the software as well as the use, study and visualization of the variables. Also it is possible to change the behaviour of the variables in real time using mathematical expressions, but these changes are not made in the code, that is to say they are momentary variations to be able to do different tests.

3.3.2.2. Python application - Python (x,y)

Python(x, y) is a free scientific and engineering development software for numerical calculations, data analysis and visualization based on the Python programming language, Qt user graphical interfaces and the interactive scientific development environment Spyder.

In [2] was already implemented a python application that allowed visualization of parameters by WiFi. This application facilitates the study of the parameters at a more advanced level, since it allows receiving data of the vehicle when it is in movement without needing to be connected to the computer, as well as to send orders to him. It is a very versatile application since it can be modified according to the needs as adding other parameters or graphics modifying the code.

4. Signal conditioning

As previously mentioned, to implement the derivative controller the signal emitted by the accelerator is used directly, so it must be conditioned in order to obtain good results.

For this, a low-pass filter has been designed to avoid possible spikes and erroneous values, and also an axes rotation system to always obtain the desired signal.

4.1. Low-pass filter design

The response will just be better as long as there is no noise signal, in case of having noise the derivative controller will amplify it, that is why a first order filter has been implemented.

As discussed above, the data obtained by the accelerometer come from variations of its environment, therefore not only detects acceleration changes on the board, which is the useful information, but that part of the data obtained are erroneous.

The signal known as the noise signal is the one that gives these erroneous data. In the electrical and electronic field, noise is considered to all electrical disturbances that interfere with the transmitted or processed signals. An irregular signal complicates the implementation of many operations and on the other hand, being in motion this can give amplified values.

The magnitude of the noise that detects the accelerometer can be checked by taking data for a certain time, keeping it as quiet as possible. Then the value of this signal is shown graphically for the three axes.

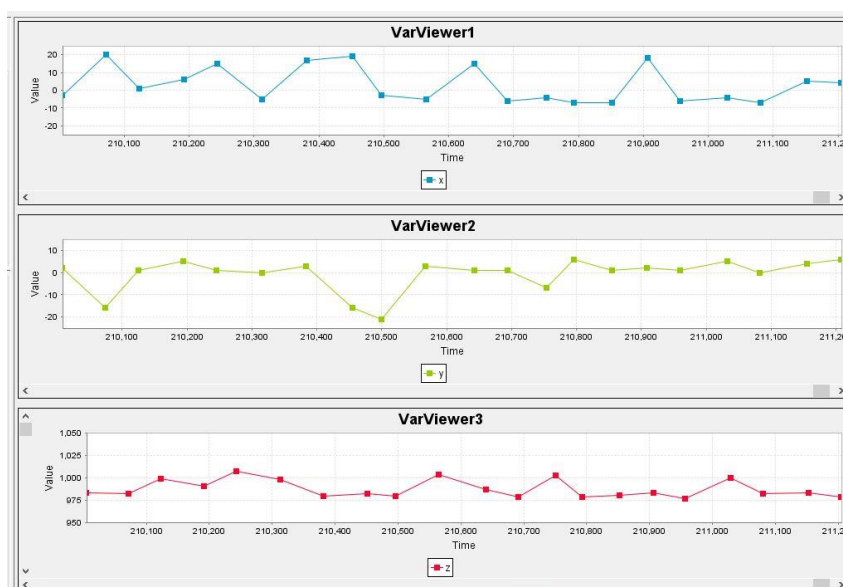


Figure 25 – Accelerometer data without filter

The accelerometer works in mg so that at rest only the force of gravity appears on the z axis around $1000mg$ and in sight, it can be seen that the value of this signal does not seem significant. Knowing that the measurement range is between $\pm 2000mg$ and the noise ranges in a range of $30mg$ the signal, it can be analytically verified that it only represents 0.75% of the signal. This may mean that the noise is due to the sampling time of the accelerometer itself.

To obtain a cleaner signal, a discrete low-pass filter is implemented, which eliminates the highest frequency signals. These types of filters have the following iterative form

$$G(s) = \frac{U(s)}{V(s)} = \frac{\omega_0}{s + \omega_0} \quad (25)$$

$$(s + \omega_0) \cdot U(s) = \omega_0 \cdot V(s) \quad (26)$$

Where $U(s)$ is the output signal and $V(s)$ the input signal of the parameter is desired to filter, in this case will be the acceleration.

Applying the inverse transform of Laplace the continuous form in time is obtained as shown below

$$\frac{du}{dt} + \omega_0 \cdot u(t) = \omega_0 \cdot v(t) \quad (24)$$

The device does not work continuously, that is, it does not collect data continuously but rather it does it every so often. For this reason, it is necessary to go from continuous time to discrete time applying the definition of the derivative, with which the following model is obtained

$$\frac{u_{k+1} - u_k}{T_s} = \omega_0 \cdot [v_k - u_k] \quad (28)$$

$$u_{k+1} = u_k \cdot [1 - T_s \cdot \omega_0] + v_k \cdot T_s \cdot \omega_0$$

In it, the parameter T_s is the signal period of the accelerometer, known as the sampling time, which is the time between each data collection and ω_0 is the cut-off frequency so, therefore the maximum frequency that the filter will allow to pass.

The noise signal does not follow any regular model from which its period can be obtained directly, then a series of possible periods are approximated graphically with which the means will be made to determine this value. It is determined to the three axes, a frequency of noise between 8 and 10 Hz, therefore ω is around 50rad/s.

In order to determine the cut-off frequency of the filter, a spectral analysis must be carried out with which the highest frequencies that are to be eliminated by the filter are obtained.

The sampling time of the accelerometer is $1ms$ but in order to pick up the data, one of the two viewers must be used that have another sampling time. This means that the data obtained finally

would have been sampled twice at different times, which is not correct and although a sampling was imposed on the viewers equal to that of the accelerometer, they would not have to be synchronous so it would continue to be Incorrect.

Consequently, a more experimental method has been used to make a sampling of the data using an oscilloscope and a Dac at a lower speed ($50\mu s$) than that of sampling ($1ms$). The answer obtained for the x axis noise at rest is as follows

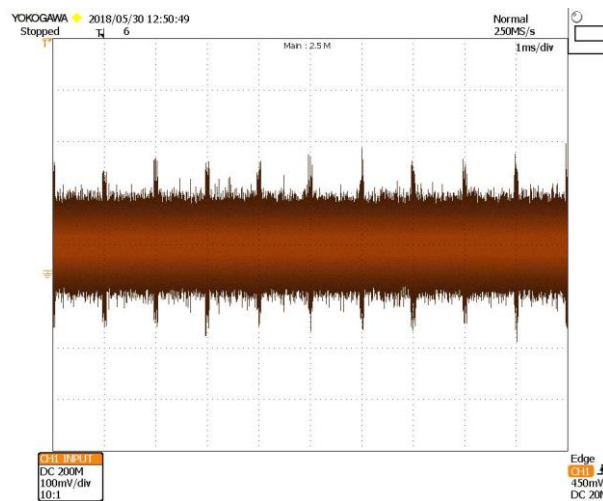


Figure 26 – Response trough the Dac

The signal amplitude (y axis) is somewhat above $100mV$, a very small value compared to the measurement range of the accelerometer which is between $\pm 2V$. The time division (x axis) is $1ms$ where a peak corresponding to the accelerometer data jack is observed. With this, it can already be verified that the noise comes from the accelerometer itself.

A FFT spectral analysis has been done with the oscilloscope trying to produce a sinusoidal signal (this signal has been done manually so it is an approximation since it does not follow a correct sinew form).

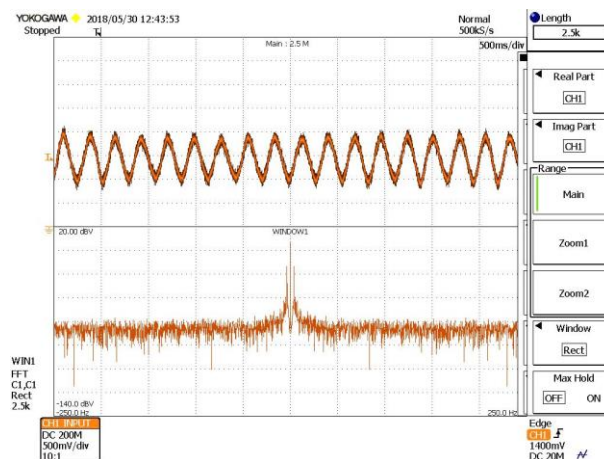


Figure 27 – Spectral Analysis in the oscilloscope

The bottom screen shows the two frequency spectrum of the signal centred in the middle of the screen and with a frequency division (x axis) of 2.5kHz. The centre peak is due to the continuous signal and as it has been supposed previously, there is no significant noise signal, not even the sampling one that is 1kHz (sampling of 1ms). All other signals are due to white noise and that the signal is an approximation.

With all this, it has been proposed to also implement the filter to work with cleaner signals and eliminate the small noise set around 50Hz. A cut-off frequency of 10rad/s has been proposed and as it is known that the data collection period T_s is 1ms, the algebraic shape of the filter is as follows

$$\begin{aligned}x_{k+1} &= (1 - 0.01) \cdot x_k + 0.01 \cdot x \\y_{k+1} &= (1 - 0.01) \cdot y_k + 0.01 \cdot y \\z_{k+1} &= (1 - 0.01) \cdot z_k + 0.01 \cdot z\end{aligned}\tag{29}$$

Where x , y and z are the values obtained by the accelerometer and the values with subscript k and $k+1$ are the previous and current filtered values.

An important addition is that for the data obtained in the y and z axis of the accelerometer the same previous study has been done, obtaining very similar results to those of the x axis, for that reason the same filter has been implemented to the three axes

Once the filter is applied to the oscilloscope, a small reduction in the amplitude of the signal is observed.

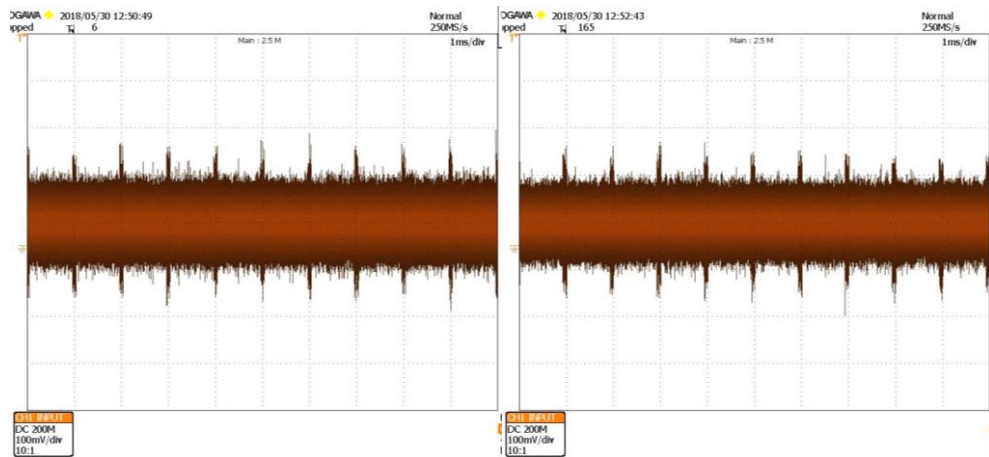


Figure 28– Comparison of the response with the Dac without using filter (left) and with it (right)

Although it seems a change almost non-existent, below is shown by the viewer the improvement of the response to each axis when it is at rest and when a movement is made wherever x , y and z are the unfiltered signals and x_{k+1} , y_{k+1} and z_{k+1} are the leaked signals.

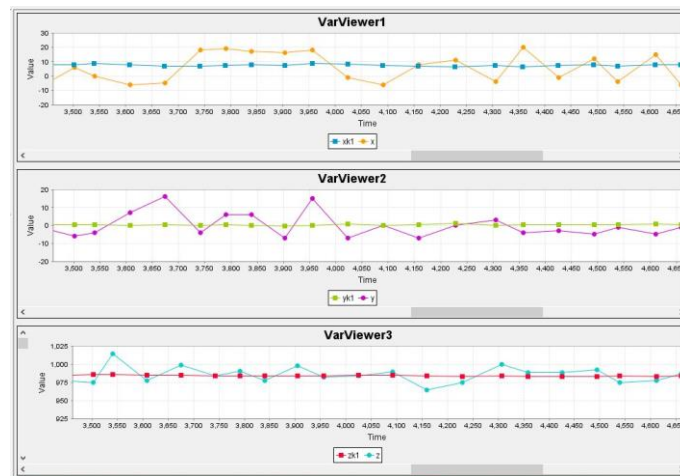


Figure 29 - Comparison of the filtered and unfiltered signals at rest

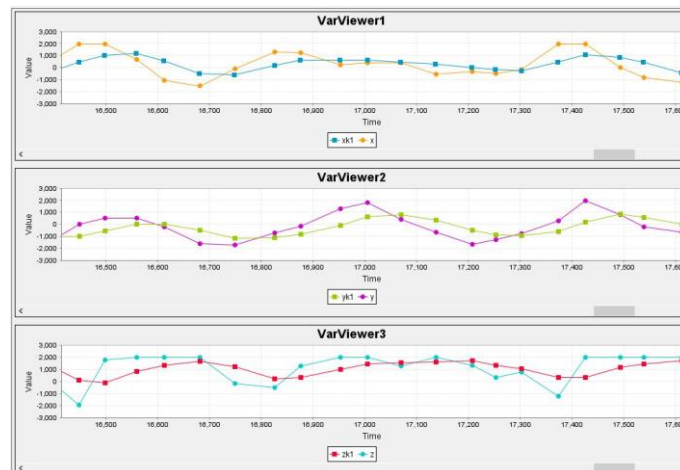


Figure 30 - Comparison of the filtered and unfiltered signals in motion

4.2. Axes rotation system

For the study carried out, it is necessary that the x and y axes are always parallel to the ground, or in other words, that the axis z is always perpendicular to the ground. Therefore, as it is assumed that the vehicle will only move parallel to the ground (without changes of height) the z acceleration component must constantly show the action of gravity ($1g$), the component y the acceleration rectilinear and the component x will appear when rotating (it also appears if there is movement perpendicular to the axis and but it is not taken into account since the vehicle cannot make this movement).

As the accelerometer itself has determined the orientation of the axes, as shown in section 3.1.1, when coupling the board to the vehicle due to irregularities in contact and ground surfaces, these will not be well-oriented. To solve this, a rotation of the axes is carried out.

Below is how the real axis and the accelerometer axis would be available when the board is oriented in different ways.

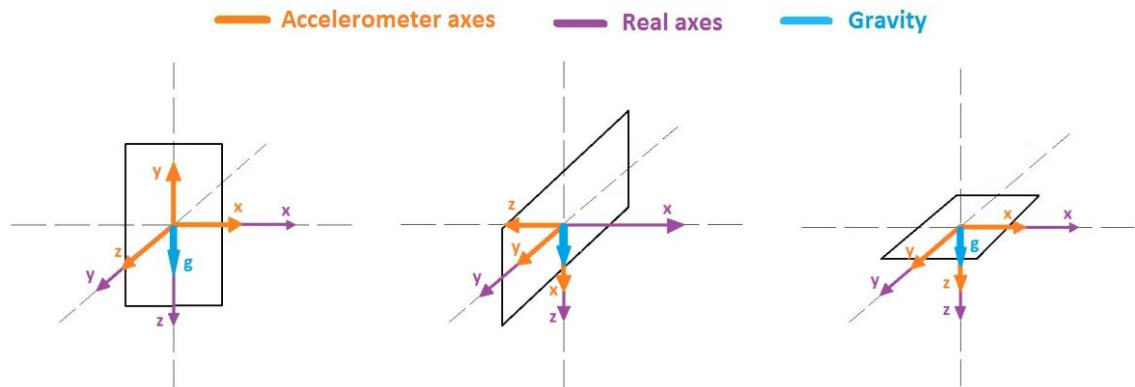


Figure 31 – Orientation of the accelerometer axes and the real axes

The rotation system used is very common in navigation systems and is based on the rotation independent roll, pitch and yaw of the axes x , y and z respectively. This system determines that the orientation of an object with respect to the axes of the earth, can be described according to a rotation of each axis x , y and z .

In this way, once the board is attached to the vehicle and in a state of rest, it can be determined which deviation have the axes of the accelerometer relative to those of the earth.

The rotation on the z axis is not considered since the imposed condition is that this axis is linear with gravity, and it is assumed that the orientation of the x and y axis once coupled the plate to the vehicle, is correct. The reference of the rotation system is therefore the component of gravity since it is always constant.

In the first place the angles of rotation with respect to axis x and y are determined when the vehicle is at rest, since thus all the acceleration that detects the accelerometer (x_g, y_g, z_g) in any of the three axes will be part of the gravity.

$$g = \sqrt{x_g^2 + y_g^2 + z_g^2}$$

$$\phi = \tan^{-1} \left(\frac{y_g}{z_g} \right)$$

$$\theta = \tan^{-1} \left(\frac{x_g}{\sqrt{y_g^2 + z_g^2}} \right)$$

$$\psi = 0$$

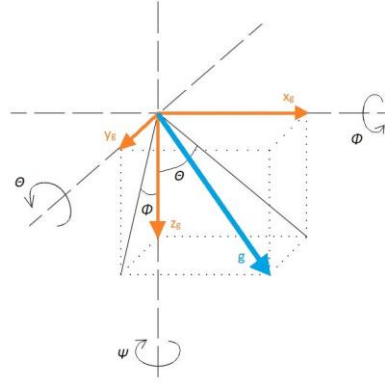


Figure 32 – Acceleration decomposition

Once these parameters have been determined, the change of axes can be carried out. It has been taken into account that in motion, the accelerometer detects both the acceleration, which is the part that is wanted to be obtained, and the action of gravity therefore, it must be eliminated.

Accelerations in motion captured by the accelerometer $\begin{Bmatrix} x \\ y \\ z \end{Bmatrix}$

Acceleration at rest $\begin{Bmatrix} x_g \\ y_g \\ z_g \end{Bmatrix}$

Desired accelerations $\begin{cases} x' = x - x_g \\ y' = y - y_g \\ z' = z - z_g \end{cases}$

It has been decided to make the rotation in order x, y, z , therefore the rotation matrix $R = R_x \cdot R_y \cdot R_z$ remains as follows:

$$R = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ \cos \psi \sin \theta \sin \phi - \cos \phi \sin \psi & \cos \phi \cos \psi + \sin \theta \sin \phi \sin \psi & \cos \theta \sin \phi \\ \cos \phi \cos \psi \sin \theta + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \cos \psi \sin \phi & \cos \theta \cos \phi \end{bmatrix} \quad (30)$$

Therefore, the system that determines the final accelerations after the filter has passed, eliminated the gravity component and correctly oriented, is as follows:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = R \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \Big|_{\psi=0} \equiv \begin{pmatrix} x' \cdot \cos \theta - z' \cdot \sin \theta \\ x' \cdot \sin \theta \sin \phi + y' \cdot \cos \phi + z' \cdot \cos \theta \sin \phi \\ x' \cdot \cos \phi \sin \theta - y' \cdot \sin \phi + z' \cdot \cos \theta \cos \phi \end{pmatrix} \quad (31)$$

5. Experimental tests

The last part of this work is that once the gains have been determined and the controller has been implemented, the following circuit is followed to see how the robot really responds.

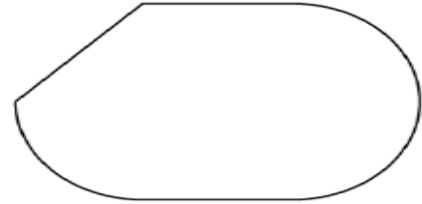
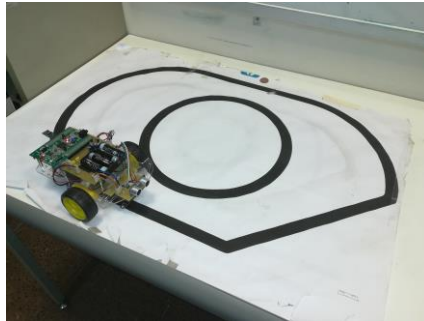


Figure 33 – Path of the circuit

As discussed above, the real answer does not have to look like the simulated since there are many parameters of the robot that are not taken into account.

Two tests have been done giving random values to K_P and K_D and a last one with the values obtained theoretically in section 2.3.2. These values are the following:

TEST	K_P	K_D
1	1	2
2	2	5
3	19	24

TABLE 2 - GAINS IN EACH TEST

The first parameter studied in these tests has been the distance between the sensor and the line of the circuit d , determining the error that is made by varying the gains.

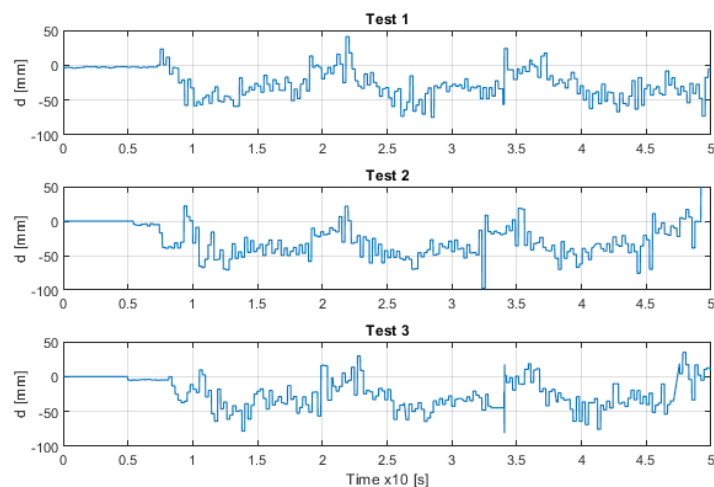


Figure 34 – Response for parameter d

The graph clearly shows the pattern of each round and with very similar values between each test. The fact that for different values of the gains this parameter varies so little is not the answer that was expected and suggests that something does not work as it should.

Then the angular speed of the vehicle has been studied since on one hand the angular speed desired by the controller is known and on the other hand, by means of the kinematic relation (2) the actual speed at which the vehicle is operating can be calculated.

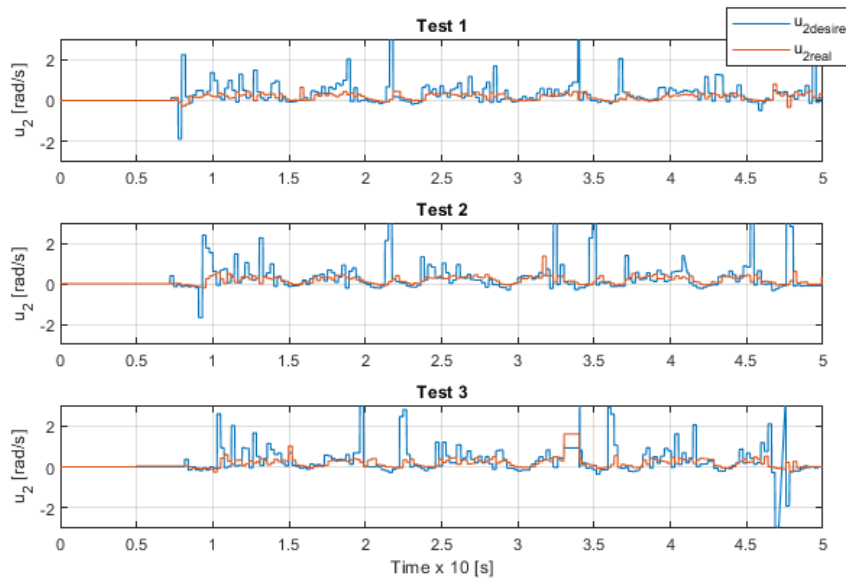


Figure 35 – Response for parameter u_2

In this case, an expected response is obtained, since the speed desired by the controller is an ideal speed that does not take into account frictions and external factors that affect the real speed, making it always lower. In addition, it can be observed that the real speed is within the working range determiner in section 2.3.2.

CONCLUSIONS

The conclusions of this work have not been all expected. It has worked with a very simplified model which is affected by different factors to which the supply of work did not reach, therefore the response of the controller has not been the ideal.

Still, another of the objectives of the work was the theoretical design of a derivative controller that directly uses the acceleration detected by the accelerometer, which has been carried out throughout the study of the system and once achieved has been validated by simulations in Matlab, where it has been observed that the response of the system was correct and concordant with the specifications of the system.

Therefore, the echo of the implementation is left for later projects in which the factors that affect its functioning are studied.

Finally, another intermediate objective of the work, was the realization and implementation of a low pass filter to obtain a cleaner signal of the acceleration, and a system of rotation of the axes of the accelerometer. Both have been validated experimentally giving good results and proving that they were correct.

BUDGET

This work corresponds to 12 ECTS credits equivalent to 300 hours of which has been divided into 150 work in the laboratory, 100 autonomous work of learning and information search, and 50 in the preparation of documentation.

As it is an end-of-grade project, no salary has been received nor has it been necessary to purchase the necessary work devices, but taking into account that a salary has been received and that the necessary tools are not available, the budget it would be like this:

CONCEPT	UNIT RATE [€/UT]	UNITS [UT]	COST [€]
WORKING HOURS	25	300	7500
PERSONAL COMPUTER	700	1	700
MICROCONTROLLER	10	1	10
SOFTWARE	0	0	0
SUBTOTAL			8210
VAT (21%)			1724
TOTAL			9934

TABLE 3 - BREAKDOWN OF THE BUDGET

ACKNOWLEDGMENTS

I would like to thank my tutors Arnau Dòria Cerezo and Victor Repecho del Corral for all the help they have offered me to carry out this project and for paying attention to me when there has been some inpreview. Thanks to this I have learned about a subject in which I had never moved and which I have enjoyed a lot.

I would also like to thank the IOC laboratory team for the good reception I received and of course my family and friends, with whom I have been able to tell at any time and situation and with whom without them I could not have reached where I am.

BIBLIOGRAPHY

- [1] Costa Ruíz, Albert. *Design of controllers and its implementation for a line tracker vehicle*. End of Degree Project UPC (2017).
- [2] Riera Seguí, Antoni. *Disseny i implementació d'un sistema de comunicacions WiFi per a una xarxa de vehicles autònoms*. End of Degree Project UPC (2016).
- [3] Freescale Semiconductor. *Tilt Sensing Using a Three-Axis Accelerometer*. Theory about the rotation of axes of an accelerometer (2013).
- [4] STM Microelectronics. *MEMS digital output sensor: ultra-low-power high-performance three-axis 'nano' accelerometer*. MEMS datasheet (2017).
- [5] STM Microelectronics. *Discovery kit with STM32F407VG MCU*. Discovery kit user manual (2017).
- [6] STM Microelectronics. *Getting started with STM-STUDIO*. STM-STUDIO viewer user manual (2013).

ANNEXE: Software Code

```

30 * File Name : main.c
33 /* Includes -----*/
34 // #include "stm32f4xx_hal.h"
35
36 #include <stdio.h>
37 #include "stm32f4xx_hal.h"
38 #include "stm32f4_discovery.h"
39 #include <comunicacions.h>
40 #include "defines.h"
41 //inclouem la llibreria math per fer probes
42 #include <math.h>
43
44 /* Variables acceleròmetre */
45 int16_t Buffer[3];
46 __IO uint8_t UserButtonPressed = 0x00;
47 __IO float x, y, z, xk, xk1, yk, yk1, zk, zk1, xg, yg, zg, xx, yy, zz, ax, ay, az, A, B, C, day, day1, vy, vy1;
48
49 /* Acceleròmetre */
50 extern void BSP_ACCELER0_Init (void);
51 extern void BSP_ACCELER0_GetXYZ (int16_t *pDataXYZ);
52 static void accel_comp (void);
53 static void angle_comp (void);
54
55 /* Buffers comunicacio */
56
57 extern uint8_t paquetSortida[MIDA_MAXIMA_PAQUET];
58 extern BufferFIFO bufferEntradaUART;
59 uint8_t caracterRx;
60 uint8_t start = 0;
61
62 /* ##### */
63
64 /* USER CODE BEGIN Includes */
65 // **Variables que inicialitzen el programa**
66
67
68 // **SENYALS DE REFERENCIA
69
70 __IO float wRef=0.0; // velocitat que volem tenir m/s ; wRef=2*u1-wRef
71 __IO float wLref=0.0; // velocitat que volem tenir m/s ; wLref=R*u2-u1
72
73 __IO float wRefV=0.0;
74 __IO float wLrefV=0.0;
75
76
77 // **CONTROLADORS PROPORCIONAL I INTEGRAL
78
79 // senyals controladors
80
81 __IO float dutyRref; //duty en % va de 0 a 100
82 __IO float dutyLref;
83
84 __IO float errorR=0; // diferencia wmeasured amb wreferencia
85 __IO float errorL=0;
86
87 __IO float SenyIntR=0; // Senyal del control integral actuara a la right wheel
88 __IO float SenyIntL=0; // Senyal del control integral actuara a la left wheel
89
90 __IO float dutyRcorr; // duty en % va de 0 a 100 un cop aplicat senyal de control
91 __IO float dutyLcorr;
92
93 // constants dels controladors
94
95 __IO float KpMotor=70; // control proporcional, per cada motor es igual, amb 30 va be
96 __IO float KiMotor=1; // control integral, per cada motor es igual, amb 0.001 va be // KiMotor es 0,001
97
98
99 // **ENCODERS
100
101 //mesura
102
103 // __IO uint16_t uwFrequency[2];
104 __IO uint16_t uwIC2Value[2]={35000, 35000}; // comptes fetes entre dos flancs del encoder (dos valors, un per a cada roda), inicialitzem en un numero molt gran
105 __IO uint16_t PrescalerEnc=1300; // amb prescaler de 1300 obtenim la maxima resolucio podem mesurar algo mes de 1 segon de temps entre flancs
106 // __IO uint32_t u=1;
107 __IO float wLmeasured;
108 __IO float wRmeasured;
109
110 __IO float wLmeasuredV;
111 __IO float wRmeasuredV;
112
113 // w=velocitat lineal = Numd/DenW = 2*r*pi*fCLKtim23 / 20*uwIC2Value*PrescalerEnc ; 20 son els forats del encoder i 2*r es diamectre roda 0,065m.
114 __IO float fCLKtim34=84; // 84[MHz] numero de comptes que pot fer el timer del encoder per segon
115 __IO float NumW=17153095.88; // Numd=2*r*pi*fCLKtim23, sera igual per el encoder de les dues rodes
116 __IO float DenWright;// DenW=20*uwIC2Value*prescalerEnc; es calculara per a cada interrupcio (aquest cas per roda dreta)
117 __IO float DenWleft; //roda esquerra
118
119 //W = velocitat angular = w*(1/r) = w*rinv --> r=0.0325
120 __IO float Wright; // velocitat angular roda dreta
121 __IO float Wleft; // velocitat angular roda esquerra
122

```

```

123 // mesura w0
124 __IO uint16_t conti=0; // comptador per indicar roda LEFT parada
125 __IO uint16_t contr=0; // comptador per indicar roda RIGHT parada
126
127 // **ACCIO ALS MOTORS
128
129 // Valors introduïts als motors
130 __IO float PWMspeedR;
131 __IO float PWMspeedL;
132
133 // **SENSOR DETECTOR DE OBSTACLES
134 __IO float USValue = 0;
135 __IO float InvUSValue;
136 __IO float USFreq = 0;
137 __IO float USdutyCycle = 0;
138 __IO uint16_t USLimit=61; //61% of the duty at 1,315kHz =465us
139 __IO float USd = 0;
140
141
142 // **SENYALS DEL CONTROL DE LINIA**
143 __IO uint16_t uhaADCxConvertedValue[2]; // aquesta variable es un nombre de 12 bits [0-4096] equivalent als Volts [0-3V] que trasnmeten els dos sensors (3V
144 __IO float Vdoff;
145 __IO float d_measured;
146 __IO float d_measuredV;
147
148 __IO float u1=0.18; //velocitat de creuer
149 __IO float u2; //velocitat de gir
150
151 __IO float errorD=0.0, errorD_ant=0.0;
152
153 // Controlador Proporcional Distancia
154 __IO float senyPropD=0.0;
155 __IO float Kp_dist=24; // tenint en compte el canvi d'unitats -- abans 0.3 pero en canviar la calibracio del KP això canvia per linia de 14mm //
156
157 // Controlador Derivatiu Acceleracio
158 __IO float senyDerA=0.0;
159 __IO float Kd_acc=19; // en sistema internacional
160
161 //Controlador Integral Distancia
162 __IO float senyIntD=0.0;
163 __IO float Ki_dist=0; //Ki_dist=0.000001, (0.000001 VA BE)
164
165 // temps comunicacions
166 __IO uint32_t crono=0;
167
168 /* USER CODE END Includes */
169
170
171 /* Private variables -----*/
172 ADC_HandleTypeDef hadc1;
173 DMA_HandleTypeDef hdma_adc1;
174 TIM_HandleTypeDef htim1;
175 TIM_HandleTypeDef htim3;
176 TIM_HandleTypeDef htim4;
177 TIM_HandleTypeDef *htim30, *htim40;
178 TIM_HandleTypeDef htim10;
179 TIM_HandleTypeDef htim11;
180 TIM_HandleTypeDef htim12;
181 DAC_HandleTypeDef hdac;
182
183
184 //UART_HandleTypeDef huart4;
185 //
186
187 /* USER CODE BEGIN PV */
188 /* Private variables -----*/
189
190 /* USER CODE END PV */
191
192
193 /* Private function prototypes -----*/
194 void SystemClock_Config(void);
195 static void MX_GPIO_Init(void);
196 static void MX_DMA_Init(void);
197 static void MX_ADC1_Init(void);
198 static void MX_TIM1_Init(void);
199 static void MX_TIM3_Init(void);
200 static void MX_TIM4_Init(void);
201 static void MX_TIM10_Init(void);
202 static void MX_TIM11_Init(void);
203 static void MX_TIM12_Init(void);
204
205 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
206
207 /* USER CODE BEGIN PFP */
208 /* Private function prototypes -----*/
209
210 /* USER CODE END PFP */
211
212 /* USER CODE BEGIN 0 */
213
214 int main(void)
215 {

```

```

216  /* USER CODE BEGIN 1 */
217
218  /* USER CODE END 1 */
219
220  /* MCU Configuration-----*/
221
222  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
223  HAL_Init();
224
225  /* Configure the system clock */
226  SystemClock_Config();
227
228  /* SysTick end of count event each 10ms */
229  // SystemCoreClock = HAL_RCC_GetHCLKFreq();
230  //SysTick_Config(SystemCoreClock / 100);
231
232  /* Inicialitzar acceletrometr */
233  BSP_ACCELERO_Init();
234
235  /* Initialize all configured peripherals */
236
237  BSP_LED_Init(LED3);
238  BSP_LED_Init(LED4);
239  BSP_LED_Init(LED5);
240  BSP_LED_Init(LED6);
241
242  MX_GPIO_Init();
243  MX_DMA_Init();
244  MX_ADC1_Init();
245
246  MX_TIM1_Init();
247  MX_TIM3_Init();
248  MX_TIM4_Init();
249  MX_TIM10_Init();
250  MX_TIM11_Init();
251  MX_TIM12_Init();
252  MX_DAC_Init();
253
254  //
255  HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_1); HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_1);
256  HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_2); HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_2);
257  HAL_TIM_PWM_Start(&htim10,TIM_CHANNEL_1);
258  /* USER CODE END 2 */
259  //tocado para capture mode
260  HAL_TIM_IC_Start(&htim11, TIM_CHANNEL_1);
261  HAL_TIM_IC_Start(&htim12, TIM_CHANNEL_1);
262  HAL_TIM_IC_Start(&htim12, TIM_CHANNEL_2);
263
264  HAL_TIM_IC_Start(&htim3, TIM_CHANNEL_1);
265  HAL_TIM_IC_Start(&htim4, TIM_CHANNEL_2);
266
267  HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
268
269  //fin tocado
270
271  HAL_Delay(1000); //Esperar que arranqui l'esp8266 o saltaran errors del port UART
272
273  // **INICIALIZACIO DEL SOFTWARE DE COMUNICACIONES**
274  BSP_LED_On(LED6);
275  inicialitzarUart();
276  //USART2->BRR = 0x16C;
277  rebreUartIT(&caracterRx, 1);

```

```

configurarConnexioWifi();
establirConnexio();
HAL_Delay(100);
inicialitzaBufferFIFO(&bufferEntradaUART); /* Una altra vegada */
BSP_LED_Off(LED6);

//HAL_GPIO_WritePin(GPIOB,GPIO_PIN_13,GPIO_PIN_RESET)

// **INFINITE LOOP*/
while (1)
{
//  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
//  COM_gestionarComunicacions();
//  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
//  if (BSP_PB_GetState(BUTTON_KEY) != KEY_NOT_PRESSED)
//  {
//      HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
//      angle_comp();
//      HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
//  }
}

}

303 void HAL_SYSTICK_Callback(void)
304 {
305     HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
306     BSP_ACCELERO_GetXYZ(Buffer);
307     x = Buffer[0];
308     y = Buffer[1];
309     z = Buffer[2];
310     /* Filtre x */
311     xk1 = (float) (1-(TS_W0X))*xk + (x*TS_W0X);
312     xk=xk1;
313     /* Filtre y */
314     yk1 = (float) (1-(TS_W0Y))*yk + (y*TS_W0Y);
315     yk=yk1;
316     /* Filtre z */
317     zk1 = (float) (1-(TS_W0Z))*zk + (z*TS_W0Z);
318     zk=zk1;
319
320     accel_comp();
321
322     HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
323 }
324
325 void angle_comp (void)
326 {
327     yg = yk1;
328     zg = zk1;
329     xg = xk1;
330     A = atan( yg / zg );
331     B = atan( xg / (sqrt( pow(yg, 2) + pow(zg, 2) )));
332     C = 0;
333 }

```



```

335 void accel_comp (void)
336 {
337     xx = xk1 - xg;
338     yy = yk1 - yg;
339     zz = zk1 - zg;
340     ax = xx*cos(B)*cos(C) + yy*cos(B)*sin(C) - zz*sin(B);
341     ay = xx*( cos(C)*sin(B)*sin(A) - cos(A)*sin(C) ) + yy*( cos(A)*cos(C) + sin(B)*sin(A)*sin(C) ) + zz*cos(B)*sin(A);
342     az = xx*( cos(A)*cos(C)*sin(B) + sin(A)*sin(C) ) + yy*( cos(A)*sin(B)*sin(C) - cos(C)*sin(A) ) + zz*cos(B)*cos(A);
343
344     /* Controlador derivatiu de l'acceleracio */
345     vy1=TS*ay*0.0098+vy;
346     day1=-(float)(ld/la)*vy1;
347     senyDerA=(float)Kd_acc*day1;
348     vy=vy1;
349 }
350
351
352 /** System Clock Configuration
353 */
354 void SystemClock_Config(void)
355 {
356     RCC_OscInitTypeDef RCC_OscInitStruct;
357     RCC_ClkInitTypeDef RCC_ClkInitStruct;
358
359     __PWR_CLK_ENABLE();
360
361     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
362
363     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
364
365     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
366     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
367     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
368     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
369     RCC_OscInitStruct.PLL.PLLM = 4;
370     RCC_OscInitStruct.PLL.PLLN = 168;
371     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
372     RCC_OscInitStruct.PLL.PLLQ = 4;
373     HAL_RCC_OscConfig(&RCC_OscInitStruct);
374
375     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
376     |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
377     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
378     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
379     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
380     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
381     HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
382
383     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/10);
384
385     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
386
387     /* SysTick_IRQn interrupt configuration */
388     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
389 }
390 /* ADC1 init function */
391 void MX_ADC1_Init(void) //capta els volts del sensor de linia
392 {

```

```

392 ADC_ChannelConfTypeDef sConfig;
393
394 /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
395 */
396 hadc1.Instance = ADC1;
397 hadc1.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV2;
398 hadc1.Init.Resolution = ADC_RESOLUTION12b;
399 hadc1.Init.ScanConvMode = ENABLE;
400 hadc1.Init.ContinuousConvMode = DISABLE;
401 hadc1.Init.DiscontinuousConvMode = DISABLE;
402 hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
403 hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T1_CC1;
404 hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
405 hadc1.Init.NbrOfConversion = 2;
406 hadc1.Init.DMAContinuousRequests = ENABLE;
407 hadc1.Init.EOCSelection = DISABLE;
408 HAL_ADC_Init(&hadc1);
409
410 /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
411 */
412 sConfig.Channel = ADC_CHANNEL_8;
413 sConfig.Rank = 1;
414 sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
415 HAL_ADC_ConfigChannel(&hadc1, &sConfig);
416
417 /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
418 */
419 sConfig.Channel = ADC_CHANNEL_9;
420 sConfig.Rank = 2;
421 sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
422 sConfig.Offset = 0;
423 HAL_ADC_ConfigChannel(&hadc1, &sConfig);
424 }
425 /* TIM1 init function */
426 void MX_TIM1_Init(void) //per la interrupcio i controlar els motors
427 {
428     TIM_ClockConfigTypeDef sClockSourceConfig;
429     TIM_MasterConfigTypeDef sMasterConfig;
430     TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig;
431     TIM_OC_InitTypeDef sConfigOC;
432
433     htim1.Instance = TIM1;
434     htim1.Init.Prescaler = 0;
435     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
436     htim1.Init.Period = 8400;
437     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
438     htim1.Init.RepetitionCounter = 0;
439     HAL_TIM_Base_Init(&htim1);
440
441     //the modes of both channels have to be opposite to turn the wheels on the same sense
442     sConfigOC.OCMode = TIM_OCMODE_PWM1;
443     sConfigOC.Pulse = 1200;
444     sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
445     sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
446     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
447     sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
448     sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
449     HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1);
450

```

```

451     sConfigOC.OCMode = TIM_OCMode_PWM2;
452     sConfigOC.Pulse = 1200;
453     HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2);
454
455     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
456     HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig);
457
458     HAL_TIM_PWM_Init(&htim1);
459
460     sMasterConfig.MasterOutputTrigger = TIM_TRGO_OC1; //TIM_TRGO_UPDATE // TIM_TRGO_ENABLE ;
461     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
462     HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig);
463
464
465     sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
466     sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
467     sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
468     sBreakDeadTimeConfig.DeadTime = 0;
469     sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
470     sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
471     sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_ENABLE;
472     HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig);
473
474     HAL_TIM_MspPostInit(&htim1);
475 }
476 /* TIM3 init function */
477 //ENCODER RIGHT WHEEL
478 void MX_TIM3_Init(void) //per encoder essequce
479 {
480
481     TIM_ClockConfigTypeDef sClockSourceConfig;
482     TIM_MasterConfigTypeDef sMasterConfig;
483     TIM_IC_InitTypeDef sConfigIC;
484     TIM_SlaveConfigTypeDef sSlaveConfig;
485
486     htim3.Instance = TIM3;
487     htim3.Init.Prescaler = PrescalerEnc;
488     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
489     htim3.Init.Period = 65535; //htim3.Init.Period = 65535;
490     htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
491     HAL_TIM_Base_Init(&htim3);
492
493     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
494     HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig);
495
496     HAL_TIM_IC_Init(&htim3);
497
498     //ojo aqui has cambiado cosas:
499     //inicial:
500     //sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
501     //sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
502     sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET; //TIM_SLAVEMODE_TRIGGER; //TIM_SLAVEMODE_RESET;
503     sSlaveConfig.InputTrigger = TIM_TS_TTFP1;
504     sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
505     sSlaveConfig.TriggerFilter = 0;
506     HAL_TIM_SlaveConfigSynchronization(&htim3, &sSlaveConfig);
507
508     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
509     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
510
511     HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig);
512
513     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
514     sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
515     sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
516     sConfigIC.ICFilter = 0;
517     HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_1);
518
519     //TIM3->CR1 = TIM_CR1_UDIS;
520
521 }
522 /* TIM4 init function */
523 //ENCODER LEFT WHEEL
524 void MX_TIM4_Init(void) //per encoder dret
525 {
526
527     TIM_ClockConfigTypeDef sClockSourceConfig;
528     TIM_MasterConfigTypeDef sMasterConfig;
529     TIM_IC_InitTypeDef sConfigIC;
530     TIM_SlaveConfigTypeDef sSlaveConfig;
531
532     htim4.Instance = TIM4;
533     htim4.Init.Prescaler = PrescalerEnc;
534     htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
535     htim4.Init.Period = 65535; //htim4.Init.Period = 65535;
536     htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
537     HAL_TIM_Base_Init(&htim4);
538
539     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
540     HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig);
541
542     HAL_TIM_IC_Init(&htim4);
543

```

```

544 //igual que en tim3
545 sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
546 sSlaveConfig.InputTrigger = TIM_TS_TI2FP2;
547 sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
548 sSlaveConfig.TriggerFilter = 0;
549 HAL_TIM_SlaveConfigSynchronization(&htim4, &sSlaveConfig);
550
551 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
552 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
553 HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig);
554
555 sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
556 sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
557 sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
558 sConfigIC.ICFilter = 0;
559 HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_2);
560
561 //TIM4->CR1 = TIM_CR1_UDIS;
562
563 }
564 /* TIM10 init function */
565 void MX_TIM10_Init(void)
566 {
567     TIM_OC_InitTypeDef sConfigOC;
568
569     htim10.Instance = TIM10;
570     htim10.Init.Prescaler = 1;
571     htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
572     htim10.Init.Period = 63900;
573     htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
574     HAL_TIM_Base_Init(&htim10);
575     HAL_TIM_Base_Init(&htim10);
576     HAL_TIM_PWM_Init(&htim10);
577
578     sConfigOC.OCMode = TIM_OCMODE_PWM1;
579     sConfigOC.Pulse = 859;
580     sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
581     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
582     HAL_TIM_PWM_ConfigChannel(&htim10, &sConfigOC, TIM_CHANNEL_1);
583
584     HAL_TIM_MspPostInit(&htim10);
585 }
586 /* TIM11 init function */
587 void MX_TIM11_Init(void)
588 {
589     TIM_IC_InitTypeDef sConfigIC;
590
591     htim11.Instance = TIM11;
592     htim11.Init.Prescaler = 140;
593     htim11.Init.CounterMode = TIM_COUNTERMODE_UP;
594     htim11.Init.Period = 65535;
595     htim11.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
596     HAL_TIM_Base_Init(&htim11);
597
598     HAL_TIM_IC_Init(&htim11);
599
600
601     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
602     sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
603     sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
604     sConfigIC.ICFilter = 0;

```

```

605 HAL_TIM_IC_ConfigChannel(&htim11, &sConfigIC, TIM_CHANNEL_1);
606 }
607 /* TIM12 init function */
608 void MX_TIM12_Init(void)
609 {
610     TIM_SlaveConfigTypeDef sSlaveConfig;
611     TIM_IC_InitTypeDef sConfigIC;
612     htim12.Instance = TIM12;
613     htim12.Init.Prescaler = 140;
614     htim12.Init.CounterMode = TIM_COUNTERMODE_UP;
615     htim12.Init.Period = 65535;
616     htim12.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
617     HAL_TIM_IC_Init(&htim12);
618
619     sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
620     sSlaveConfig.InputTrigger = TIM_TS_T1FP1;
621     sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
622     sSlaveConfig.TriggerPrescaler = TIM_ICPSC_DIV1;
623     sSlaveConfig.TriggerFilter = 0;
624     HAL_TIM_SlaveConfigSynchronization(&htim12, &sSlaveConfig);
625
626     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
627     sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
628     sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
629     sConfigIC.ICFilter = 0;
630     HAL_TIM_IC_ConfigChannel(&htim12, &sConfigIC, TIM_CHANNEL_1);
631
632     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
633     sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
634     sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
635     HAL_TIM_IC_ConfigChannel(&htim12, &sConfigIC, TIM_CHANNEL_2);
636 }
637
638 /**
639  * Enable DMA controller clock
640  */
641 void MX_DMA_Init(void)
642 {
643     /* DMA controller clock enable */
644     __DMA2_CLK_ENABLE();
645
646     /* DMA interrupt init */
647     HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
648     HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
649 }
650
651 /** Configure pins as
652     * Analog
653     * Input
654     * Output
655     * EVENT_OUT
656     * EXTI
657 */
658 void MX_GPIO_Init(void)
659 {
660     GPIO_InitTypeDef GPIO_InitStruct;
661
662     /* GPIO Ports Clock Enable */
663     __GPIOA_CLK_ENABLE();
664     __GPIOB_CLK_ENABLE();
665     __GPIOC_CLK_ENABLE();
666     __GPIOD_CLK_ENABLE();
667
668     /*Configure GPIO pin : PA0 */
669     GPIO_InitStruct.Pin = GPIO_PIN_0;
670     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
671     GPIO_InitStruct.Pull = GPIO_NOPULL;
672     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
673
674     /* EXTI interrupt init*/
675     HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
676     HAL_NVIC_EnableIRQ(EXTI0_IRQn);
677
678     /*Configure GPIO pin Output Level */
679     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_RESET);
680     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET);
681     //HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
682
683     /*Configure GPIO pin : PB11 */
684     GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_13|GPIO_PIN_15;
685     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
686     GPIO_InitStruct.Pull = GPIO_NOPULL;
687     GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
688     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
689 }

```

```

700 void MX_DAC_Init(void)
701 {
702     DAC_ChannelConfTypeDef sConfig;
703
704     /**DAC Initialization */
705     hdac.Instance = DAC;
706     HAL_DAC_Init(&hdac);
707
708     /**DAC channel OUT1 config */
709     sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
710     sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
711     HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_1);
712
713 }
714
715
716 /** *CODI CONTROLS */
717 // INTERRUPTIO, Execucio cada 50 microsegons el ADC ja ha pactat valors i disponibles uADCxConvertedValue[] buffer dos posicions
718 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
719 {
720     HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_SET);
721     //COM gestionar Comunicacions();
722     // cronometre
723
724     if (crono == 0xFFFFFFFF){ crono=0;}
725
726     else{crono=crono+1;}
727
728
729 // **CAPTACIO DADES** //A
730 // pin a 1 per comprobar cada quant es fa la interrupcio
731
732 // * ENCODERS
733 uwIC2Value[0]= HAL_TIM_ReadCapturedValue(&htim3,TIM_CHANNEL_1); // encoder right wheel, llegeix el numero de comptes del encoder
734 if (uwIC2Value[0] == 0){uwIC2Value[0]=30000;}
735 DenWright=20.0*uwIC2Value[0]*PrescalerEnc;
736
737 uwIC2Value[1]= HAL_TIM_ReadCapturedValue(&htim4,TIM_CHANNEL_2); // encoder left wheel
738 if (uwIC2Value[1] == 0){uwIC2Value[1]=30000;}
739 Denleft=20.0*uwIC2Value[1]*PrescalerEnc;
740
741 // * MESURA W RODES, SENTIT GIR I DETECTOR W=0
742 //RIGHT WHEEL
743 if (contrl<20000) // ha passat 1 segon sense moviment a R, si no detectem esta parat, calculem w de la roda R
744 {
745     contrl=contrl+1;
746     wRmeasured = (float)(NumW/DenWright); //uwFrequency[0]*0.065*0.314159;
747     Wright = (float) wRmeasured * rinv;
748     if (PWMSpeedR<4200){ wRmeasured=wRmeasured*(-1.0);} //PROBLEMA amb Wref=0!!! PRESUPOSEM SIGNE DE LA W SEGONS EL DUTY QUE LI ESTEM SUBMINIST
749 }
750 else
751 {wRmeasured=0.0;}
752
753 wRmeasuredV=wRmeasured*10000;
754
755 //LEFT WHEEL
756 if (contrl<20000) // ha passat 1 segon sense moviment a L, si no detectem esta parat, calculem w de la roda L
757 {
758     contrl=contrl+1;
759     wLmeasured = (float)(NumW/Denleft); //uwFrequency[1]*0.065*3.14159/10;
760     wLeft = (float) wLmeasured * rinv;
761     if (PWMSpeedL<4200){ wLmeasured=wLmeasured*(-1.0);}
762 }
763 else
764 {wLmeasured=0.0;} // detectem roda parada, aquesta sera la mesurada
765
766 wLmeasuredV=wLmeasured*10000;
767
768 //FLANCS PER CADA RODA
769 //left wheel
770 if ((__HAL_TIM_GET_FLAG(&htim4, TIM_FLAG_TRIGGER) == 1) ) //cada vegada que el encoder passa per un flanc '_HAL_TIM_GET_FLAG()' es posa 1, despres
771 {
772     //TIM_FLAG_CC1 correspon al SR-TIM_CCR1 pg 634 manual
773     //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,1);
774     HAL_TIM_CLEAR_FLAG(&htim4, TIM_FLAG_TRIGGER); // _HAL_TIM_CLEAR_FLAG(&htim4, TIM_FLAG_CC1 ); // posem a 0 per software el flag que indica
775     //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,0);
776     contrl=0;
777 }
778 //right wheel
779 if ((__HAL_TIM_GET_FLAG(&htim3, TIM_FLAG_TRIGGER) == 1) ) //cada vegada que el encoder passa per un flanc '_HAL_TIM_GET_FLAG()' es posa 1, despres
780 {
781     //TIM_FLAG_CC1 correspon al SR-TIM_CCR1 pg 634 manual
782     //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,1);
783     HAL_TIM_CLEAR_FLAG(&htim3, TIM_FLAG_TRIGGER); // _HAL_TIM_CLEAR_FLAG(&htim4, TIM_FLAG_CC1 ); // posem a 0 per software el flag que indica
784     //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,0);
785     contrl=0;
786 }
787
788 // * DETECTOR OBSTACLE
789 USValue = HAL_TIM_ReadCapturedValue(&htim12, TIM_CHANNEL_1); // valor capta el sensor de proximitat, si es 0 no detecta res, de lo contrari, calcularem la
790 InvUSValue= (float) (1 / USValue);
791 USDutyCycle = HAL_TIM_ReadCapturedValue(&htim12, TIM_CHANNEL_2) * 100 * InvUSValue; // float(HAL_TIM_ReadCapturedValue(&htim12, TIM_CHANNEL_2)) * 100 * Inv
792 USFreq = HAL_RCC_GetCLKFreq()*InvUSValue*0.0035714; //USFreq = (float)(HAL_RCC_GetCLKFreq())/2 / USValue/140;
793 USD=(USDutyCycle/USFreq)*171.5; // (float)((USDutyCycle/USFreq)*0.01715*10000);
794
795 // DISTANCIA A LINIA
796
797 Vdiff= (float)(uADCxConvertedValue[0]-uADCxConvertedValue[1]);
798
799

```

```

824 d_measured=(0.00173*Vdiff)+0.234; // nova relacio de distancia en milimetres dist=0,0001684*Vdiff-0,1309 ABANS d_measured=(0.00173*Vdiff)+0.234;
825 d_measuredV=d_measured*10000;
826
827
828 // DAC --> miraç ax i ay per anàlisis espectral --> miraç si tot soroll es del sampling o no
829 HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,(uint32_t)(axi+20));
830
831
832 // **COMPROBEM SI PAREM COTXE**
833
834 if((start==0) || ((USD<7) && (USValue != 0))) // STOP; a través de la aplicació, o objecte a distancia proxima, o sensor de proximitat no rep senyal,
835 {
836   TIM1->CCR1=(uint16_t)(4200); //left wheel
837   TIM1->CCR2=(uint16_t)(4200); //right wheel
838
839   //REINICIALITZEM (les constants dels controladors NO)
840
841   wRref=0.0;
842   wLref=0.0;
843   wRrefV=0.0;
844   wLrefV=0.0;
845
846   // No eliminem errors, vull que continui mesurant la senyal
847   //errorL=0.0;
848   //errorR=0.0;
849
850   //errorD=0.0;
851
852   SenyIntR=0.0;
853   SenyIntL=0.0;
854   PWMSpeedR=4200;
855   PWMSpeedL=4200;
856   senyPropD=0.0;
857   senyIntD=0.0;
858
859 }
860
861 // **CALCULS I EXECUCIO**
862 else {
863
864   //CONTROL DISTANCIA A LINIA PI
865
866   errorD=d_measured; // la distancia ideal de la linia es 0mm, mig linia negra, ERROR=0.0-d_measured
867
868   senyPropD=Kp_dist*errorD; // Per a les rectes, que no trencoli tant
869   senyIntD= senyIntD+(0.00001*Ki_dist*errorD); // No aporta gaire
870
871   u2=senyPropD+senyIntD+senyDerA; //canviat
872
873   //CALCUL VELOCITATS RODES
874   wLref=0.15*u2+u1; //R=0.15m es distancia entre codes en metres
875   wRref=2*u1-wLref;
876
877   wLrefV=10000*wLref;
878   wRrefV=10000*wRref;
879
880   //CONTROL RIGHT Wheel // definim una zona de velocitat zero entre 0.05 i -0.05.
881   if (wRref>0.05){ ///
882     dutyRref= (wRref+1.6103)*46.729; //model experimental planta
883   }
884   if (wRref<-0.05){
885     dutyRref= (wRref+0.5888)*39.0625; //dutyRref= (wRref+0.5888)/0.0256; AIXI EVITEM DIVISIONS
886   }
887   if ((wRref>=0.05) && (wRref<=0.05)) {
888     dutyRref= 50; //interval entre 0.05 i -0.05 m/s en que considerarem nula la velocitat, pararem el cotxe, no podrem portar el cotxe a aqu
889     wRref=0.0;
890   }
891
892   //CONTROL LEFT Wheel
893   if (wLref>0.05){
894     dutyLref= (wLref+1.6103)*46.729; // model experimental planta, a partir de consigna wLref dona el duty en %
895   }
896   if (wLref<-0.05){
897     dutyLref= (wLref+0.5888)*39.0625;
898   }
899   if ((wLref>=0.05) && (wLref<=0.05)) {
900     dutyLref= 50; //interval entre 0.05 i -0.05 m/s en que considerarem nula la velocitat, pararem el cotxe, no podrem portar el cotxe a aqu
901     wLref=0.0;
902   }
903
904   // definim els errors dels dos motors
905   errorR= wRref-wRmeasured;
906   errorL= wLref-wLmeasured;
907
908   SenyIntR= SenyIntR+(errorR*(KiMotor)*0.001); // KiMotor es 0,001
909   SenyIntL= SenyIntL+(errorL*(KiMotor)*0.001);
910
911   dutyRcorr= dutyRref+(errorR*(KpMotor)+SenyIntR);
912   // saturem la senyal de control
913   if (dutyRcorr > 100) {dutyRcorr=100;}

```



```

913         if (dutyRcorr < 0) {dutyRcorr=0;}
914         // pasem duty [0-100] a duty [0-8500]
915
916         //dutyRcorr = 100.0; per ficar velocitat maxima de la roda dreta
917
918         PWMSpeedR=dutyRcorr*85; //PWMSpeedR=(float)(dutyRcorr*(8500/100));
919
920         dutyLcorr= dutyLref+(errorL*(KpMotor)+SenyIntL);
921         // saturem la senyal de control
922         if (dutyLcorr > 100) {dutyLcorr=100;}
923         if (dutyLcorr < 0) {dutyLcorr=0;}
924         // pasem duty [0-100] a duty [0-8500]
925
926         //dutyLcorr = 100.0; per ficar velocitat maxima de la roda esquerra
927
928         PWMSpeedL=dutyLcorr*85; //PWMSpeedL=(float)(dutyLcorr*(8500/100));
929
930
931
932 // **INTRODUIM VALORS als motors**
933
934         TIM1->CCR1=(uint16_t)(PWMSpeedL); //left wheel// TIM1->CCR1=(uint16_t)(PWMSpeedL)
935         TIM1->CCR2=(uint16_t)(PWMSpeedR); //right wheel// TIM1->CCR2=(uint16_t)(PWMSpeedR);
936
937     }
938     HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_RESET);
939
940 // *END CODI CONTROLS */
941 }
942 void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
943 {
944     void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
945     {
946         BSP_LED_Toggle(LED3);
947     }
948     void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
949     {
950         BSP_LED_Toggle(LED4);
951         escriuBufferFIFO(&bufferEntradaUART, &caracterRx, 1);
952         rebreUartIT(&caracterRx, 1);
953     }
954     void HAL_UART_ErrorCallback(UART_HandleTypeDef *UartHandle)
955     {
956         BSP_LED_On(LED5);
957     }
958 }
959 /* USER CODE END 4 */
960
961 #ifdef USE_FULL_ASSERT
962 /**
963  * @brief Reports the name of the source file and the source line number
964  * where the assert_param error has occurred.
965  * @param file: pointer to the source file name
966  * @param line: assert_param error line source number
967  * @retval None
968  */
969 void assert_failed(uint8_t* file, uint32_t line)
970 {
971     /* USER CODE BEGIN 6 */
972     /* User can add his own implementation to report the file name and line number,
973      * ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
974     /* USER CODE END 6 */

```